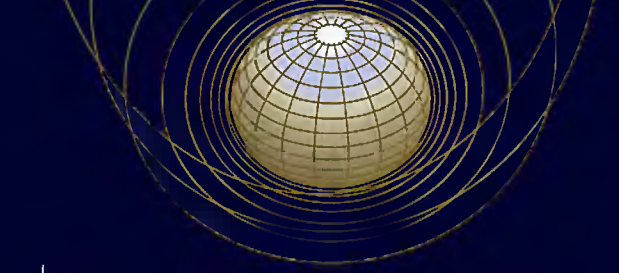
Practical Methods for Optimal Control Using Nonlinear Programming

John T. Betts



Practical Methods for Optimal Control Using Nonlinear Programming

Advances in Design and Control

SIAM's Advances in Design and Control series consists of texts and monographs dealing with all areas of design and control and their applications. Topics of interest include shape optimization, multidisciplinary design, trajectory optimization, feedback, and optimal control. The series focuses on the mathematical and computational aspects of engineering design and control that are usable in a wide variety of scientific and engineering disciplines.

Series Volumes

Betts, John T., *Practical Methods for Optimal Control Using Nonlinear Programming*

El Ghaoui, Laurent and Niculescu, Silviu-Iulian, eds., *Advances in Linear Matrix Inequality Methods in Control*

Helton, J. William and James, Matthew R., *Extending H^∞ Control to Nonlinear Systems: Control of Nonlinear Systems to Achieve Performance Objectives*

Practical Methods for Optimal Control Using Nonlinear Programming

John T. Betts
The Boeing Company
Seattle, Washington

siam
Society for Industrial and Applied Mathematics
Philadelphia

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

Betts, John T. 1943-
Practical methods for optimal control using nonlinear programming / John T. Betts.
p. cm.— (Advances in design and control)
Includes bibliographical references and index.
ISBN 0-89871-488-5
1. Control theory. 2. Mathematical optimization. 3. Nonlinear programming. I. Series.

QA402.3 .B47 2001
629.8'312--dc21

00-069809

For Theon and Dorothy

This page intentionally left blank

Contents

Preface

Solving an optimal control problem is not easy. Pieces of the puzzle are found scattered throughout many different disciplines. Furthermore, the focus of this book is on *practical methods*, that is, methods that I have found actually work! In fact everything described in this book has been implemented in production software and used to solve real optimal control problems. Although the reader should be proficient in advanced mathematics, no theorems are presented.

Traditionally, there are two major parts of a successful optimal control solution technique. The first part is the "optimization" method. The second part is the "differential equation" method. When faced with an optimal control problem, it is tempting to simply "paste" together packages for optimization and numerical integration. While naive approaches such as this may be moderately successful, the goal of this book is to suggest that there is a better way! The methods used to solve the differential equations and optimize the functions are intimately related.

The first two chapters of this book focus on the optimization part of the problem. In Chapter 1, the important concepts of nonlinear programming for small, dense applications are introduced. Chapter 2 extends the presentation to problems that are both large and sparse. Chapters 3 and 4 address the differential equation part of the problem. Chapter 3 introduces relevant material in the numerical solution of differential (and differential-algebraic) equations. Methods for solving the optimal control problem are treated in some detail in Chapter 4. Throughout the book, the interaction between optimization and integration is emphasized. Chapter 5 presents a collection of examples that illustrate the various concepts and techniques.

The book does not cover everything. Many important topics are simply not discussed in order to keep the overall presentation concise and focused. The discussion is general and presents a unified approach to solving optimal control problems. Most of the examples are drawn from my experience in the aerospace industry. Examples have been solved using a particular implementation called $\mathbb{SOCS}$. I have tried to adhere to notational conventions from both optimization and control theory whenever possible. Also, I have attempted to use consistent notation throughout the book.

The material presented here represents the collective contributions of many people. The nonlinear programming material draws heavily on the work of John Dennis, Roger Fletcher, Philip Gill, Walter Murray, Michael Saunders, and Margaret Wright. The material on differential-algebraic equations is drawn from the work of Uri Ascher, Kathy Brenan, Steve Campbell, and Linda Petzold. I was introduced to optimal control by Stephen Citron, and routinely refer to the text by Bryson and Ho [35]. Over the past 10 years I have been fortunate to participate in workshops at Oberwolfach, Munich, Minneapolis, Victoria, Lausanne, and Griefswald. I benefited immensely simply by talking with Larry Biegler, Hans Georg Bock, Roland Bulirsch, Rainer Callies, Kurt Chudej,

Tim Kelley, Bernd Kugelmann, Helmut Maurer, Rainer Mehlhorn, Angelo Miele, Hans Josef Pesch, Ekkehard Sachs, Gottfried Sachs, Roger Sargent, Volker Schulz, Mark Steinbach, Oskar von Stryk, and Klaus Well.

Two coworkers deserve special thanks. Paul Frank has played a major role in the implementation and testing of the large, sparse nonlinear programming methods described. Bill Huffman, my coauthor for many publications and the SOCS software, has been an invaluable sounding board over the last two decades. Finally, thanks to Jennifer for her patience and understanding during the preparation of this book.

John T. Betts

Chapter 1

Introduction to Nonlinear Programming

1.1 Preliminaries

This book concentrates on numerical methods for solving the optimal control problem. The fundamental principle of all effective numerical optimization methods is to solve a difficult problem by solving a sequence of simpler subproblems. In particular, the solution of an optimal control problem will require the solution of one or more finite-dimensional subproblems. As a prelude to our discussions on optimal control, this chapter will focus on the nonlinear programming (NLP) problem. The NLP problem requires finding a finite number of variables such that an *objective function* or *performance index* is optimized without violating a set of *constraints*. The NLP problem is often referred to as *parameter optimization*. Important special cases of the NLP problem include *linear programming* (LP). *quadratic programming* (QP), and *least squares* problems.

Before proceeding further, it is worthwhile to establish the notational conventions used throughout the book. This is especially important since the subject matter covers a number of different disciplines, each with their own notational conventions. Our goal is to present a unified treatment of all these fields. As a rule, scalar quantities will be denoted by lowercase letters (e.g., α). Vectors will be denoted by boldface lowercase letters and will usually be considered column vectors, as in

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \tag{1.1}$$

where the individual components of the vector are x_k for $k = 1, \ldots, n$. To save space, it will often be convenient to define the *transpose*, as in

$$\mathbf{x}^\mathsf{T} = (x_1, x_2, \ldots, x_n). \tag{1.2}$$

A *sequence* of vectors will often be denoted as $\mathbf{x}_k, \mathbf{x}_{k+1}, \ldots$. Matrices will be denoted by

boldface capital letters, as in

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}. \tag{1.3}$$

1.2 Newton's Method in One Variable

The fundamental approach to most iterative schemes was suggested over 300 years ago by Newton. In fact, Newton's method is the basis for all of the algorithms we will describe. We begin with the simplest form of Newton's method and then in subsequent sections generalize the discussion until we have presented one of the most widely used NLP algorithms, namely the *sequential quadratic programming* (SQP) method.

Suppose it is required to find the value of the variable x such that the *constraint* function

$$c(x) = 0. \tag{1.4}$$

Let us denote the solution by x^* and let us assume x is a guess for the solution. The basic idea of Newton's method is to approximate the nonlinear function $c(x)$ by the first two terms in a Taylor series expansion about the current point x. This yields a linear approximation for the constraint function at the new point $\bar{x}$, which is given by

$$c(\bar{x}) = c(x) + c'(x)(\bar{x} - x), \tag{1.5}$$

where $c'(x) = dc/dx$ is the slope of the constraint at x. Using this linear approximation, it is reasonable to compute $\bar{x}$, a new estimate for the root, by solving (1.5) such that $c(\bar{x}) = 0$, i.e.,

$$\bar{x} = x - [c'(x)]^{-1} c(x). \tag{1.6}$$

Typically, we denote $p \equiv \bar{x} - x$ and rewrite (1.6) as

$$\bar{x} = x + p, \tag{1.7}$$

where

$$p = -[c'(x)]^{-1} c(x). \tag{1.8}$$

Of course, in general, $c(x)$ is not a linear function of x, and consequently we cannot expect that $c(\bar{x}) = 0$. However, we might hope that $\bar{x}$ is a better estimate for the root x^* than the original guess x, in other words we might expect that

$$|\bar{x} - x^*| \leq |x - x^*| \tag{1.9}$$

and also

$$|c(\bar{x})| \leq |c(x)|. \tag{1.10}$$

If the new point is an improvement, then it makes sense to repeat the process, thereby defining a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \dots$ with point $(k+1)$ in the sequence given by

$$x^{(k+1)} = x^{(k)} - [c'(x^{(k)})]^{-1} c(x^{(k)}). \tag{1.11}$$

For notational convenience, it usually suffices to present a single step of the algorithm, as in (1.6), instead of explicitly labeling the information at step k using the superscript notation $x^{(k)}$. Nevertheless, it should be understood that the algorithm defines a sequence of points $x^{(0)}, x^{(1)}, x^{(2)}, \ldots$. The sequence is said to *converge* to x^* if

$$\lim_{k \to \infty} |x^{(k)} - x^*| = 0. \tag{1.12}$$

In practice, of course, we are not interested in letting $k \to \infty$. Instead we are satisfied with terminating the sequence when the computed solution is "close" to the answer. Furthermore, the *rate of convergence* is of paramount importance when measuring the computational efficiency of an algorithm. For Newton's method, the rate of convergence is said to be *quadratic* or, more precisely, *q-quadratic* (cf. [46]). The impact of quadratic convergence can be dramatic. Loosely speaking, it implies that each successive estimate of the solution will *double* the number of significant digits!

Example 1.1. To demonstrate, let us suppose we want to solve the constraint

$$c(x) = a_1 + a_2 x + a_3 x^2 = 0, \tag{1.13}$$

where the coefficients a_1, a_2, a_3 are chosen such that $c(0.1) = -0.05$, $c(0.25) = 0$, and $c(0.9) = 0.9$. Table 1.1 presents the Newton iteration sequence beginning from the initial guess $x = 0.85$ and proceeding to the solution at $x^* = 0.25$. Figure 1.1 illustrates the first three iterations. Notice in Table 1.1 that the error between the computed solution and the true value, which is tabulated in the third column, exhibits the expected doubling in significant figures from the fourth iteration to convergence.

So what is wrong with Newton's method? Clearly, quadratic convergence is a very desirable property for an algorithm to possess. Unfortunately, if the initial guess is not sufficiently close to the solution, i.e., within the *region of convergence*, Newton's method may diverge. As a simple example, Dennis and Schnabel [46] suggest applying Newton's method to solve $c(x) = \arctan(x) = 0$. This will diverge when the initial guess $|x^{(0)}| > a$, converge when $|x^{(0)}| < a$, and cycle indefinitely if $|x^{(0)}| = a$, where $a = 1.3917452002707$. In essence, Newton's method behaves well near the solution (*locally*) but lacks something permitting it to converge *globally*. So-called globalization techniques, aimed at correcting this deficiency, will be discussed in subsequent sections. A second difficulty occurs when the slope $c'(x) = 0$. Clearly, the correction defined by (1.6) is not well defined in this case. In fact, Newton's method loses its quadratic convergence property if the slope is zero at the solution, i.e., $c'(x^*) = 0$. Finally, Newton's method requires that the slope $c'(x)$ can be computed at every iteration. This may be difficult and/or costly especially when the function $c(x)$ is complicated.

1.3 Secant Method in One Variable

Motivated by a desire to eliminate the explicit calculation of the slope, one can consider approximating it at x^k by the secant

$$c'(x^k) \approx B = \frac{c(x^k) - c(x^{k-1})}{x^k - x^{k-1}} \equiv \frac{\Delta c}{\Delta x}. \tag{1.14}$$

Notice that this approximation is constructed using two previous iterations, but only requires values for the constraint function $c(x)$. This expression can be rewritten to give

Iter.	$c(x)$	x	$\lvert x - x^* \rvert$
1	0.79134615384615	0.85000000000000	0.60000000000000
2	0.18530192382759	0.47448669201521	0.22448669201521
3	$3.5942428588261 \times 10^{-2}$	0.30910437279376	$5.9104372793756 \times 10^{-2}$
4	$3.6096528286200 \times 10^{-3}$	0.25669389900972	$6.6938990097217 \times 10^{-3}$
5	$5.7007630268141 \times 10^{-5}$	0.25010744198003	$1.0744198002549 \times 10^{-4}$
6	$1.5161639596584 \times 10^{-8}$	0.25000002858267	$2.8582665845267 \times 10^{-8}$
7	$1.0547118733939 \times 10^{-15}$	0.25000000000000	$1.8873791418628 \times 10^{-15}$

Table 1.1: Newton's method for root finding.

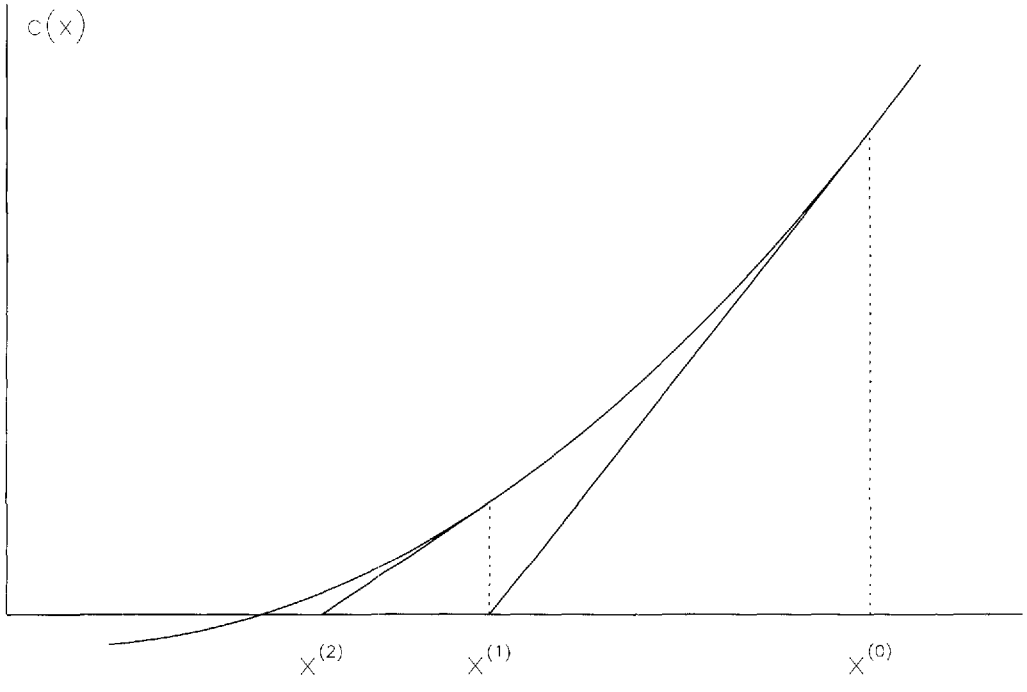


Figure 1.1: Newton's method for root finding.

the so-called secant condition

$$B\Delta x = \Delta c, \tag{1.15}$$

where B is the (scalar) secant approximation to the slope. Using this approximation, it then follows that the Newton iteration (1.6) is replaced by the secant iteration

$$\bar{x} = x - B^{-1}c(x) = x + p, \tag{1.16}$$

which is often written as

$$x^{k+1} = x^k - \frac{x^k - x^{k-1}}{c(x^k) - c(x^{k-1})}c(x^k). \tag{1.17}$$

Figure 1.2 illustrates a secant iteration applied to Example 1.1 described in the previous section.

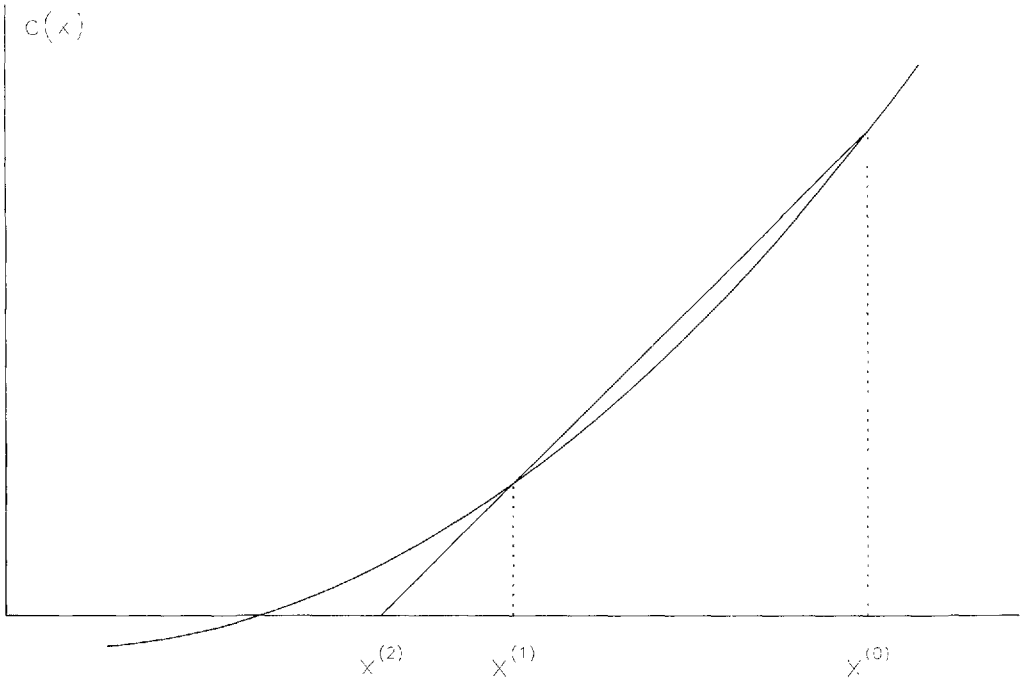


Figure 1.2: Secant method for root finding.

Clearly, the virtue of the secant method is that it does not require calculation of the slope $c'(x^k)$. While this may be advantageous when derivatives are difficult to compute, there is a downside! The secant method is *superlinearly* convergent, which, in general, is not as fast as the quadratically convergent Newton algorithm. Thus, we can expect convergence will require more iterations, even though the cost per iteration is less. A distinguishing feature of the secant method is that the slope is approximated using information from previous iterates in lieu of a direct evaluation. This is the simplest example of a so-called quasi-Newton method.

1.4 Newton's Method for Minimization in One Variable

Now let us suppose we want to compute the value x^* such that the nonlinear *objective function* $F(x^*)$ is a minimum. The basic notion of Newton's method for root finding is to approximate the nonlinear constraint function $c(x)$ by a simpler model (i.e., linear) and then compute the root for the linear model. If we are to extend this philosophy to optimization, we must construct an approximate model of the objective function. Just as in the development of (1.5), let us approximate $F(x)$ by the first three terms in a Taylor series expansion about the current point x:

$$F(\bar{x}) = F(x) + F'(x)(\bar{x} - x) + \frac{1}{2}(\bar{x} - x)F''(x)(\bar{x} - x). \tag{1.18}$$

Notice that we cannot use a linear model for the objective because a linear function does not have a finite minimum point. In contrast, a quadratic approximation to $F(x)$ is the

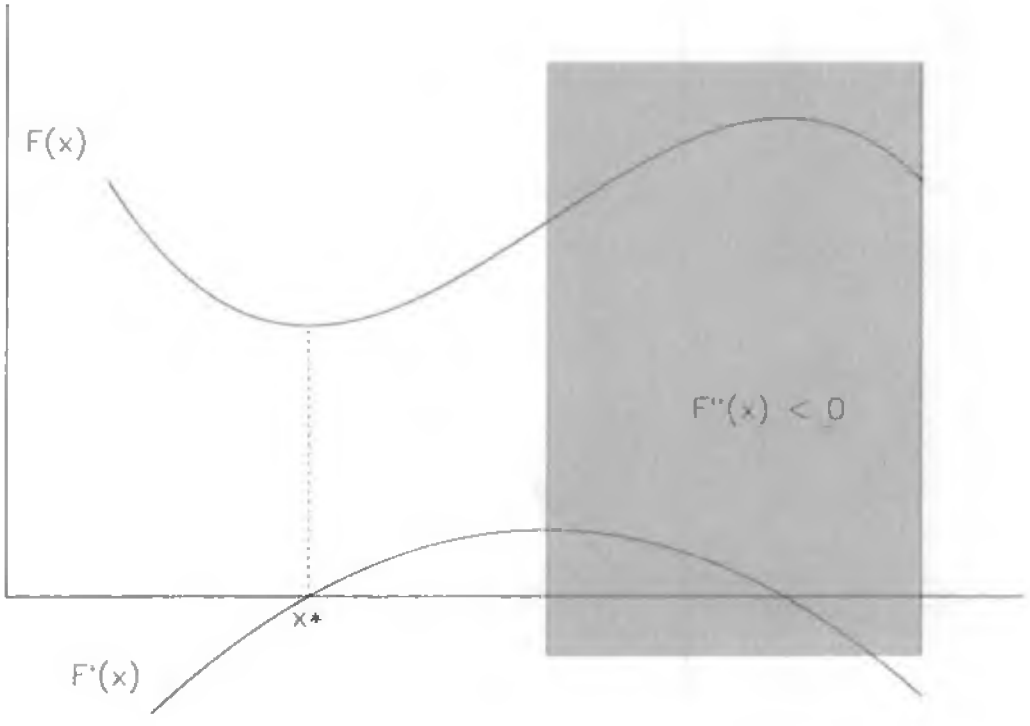


Figure 1.3: Minimization in one variable.

simplest approximation that does have a minimum. Now for $\bar{x}$ to be a minimum of the quadratic (1.18), we must have

$$\frac{dF}{d\bar{x}} \equiv F'(\bar{x}) = 0 = F'(x) + F''(x)(\bar{x} - x). \tag{1.19}$$

Solving for the new point yields

$$\bar{x} = x - [F''(x)]^{-1} F'(x). \tag{1.20}$$

The derivation has been motivated by minimizing $F(x)$. Is this equivalent to solving the slope condition $F'(x) = 0$? It would appear that the iterative *optimization* sequence defined by (1.20) is the same as the iterative *root-finding* sequence defined by (1.6), provided we replace $c(x)$ by $F'(x)$. Clearly, a quadratic model for the objective function (1.18) produces a linear model for the slope $F'(x)$. However, the condition $F'(x) = 0$ only defines a *stationary point*, which can be either a minimum, a maximum, or a point of inflection. Apparently what is missing is information about the *curvature* of the function, which would determine whether it is concave up, concave down, or neither.

Figure 1.3 illustrates a typical situation. In the illustration, there are two points with zero slopes; however, there is only one minimum point. The minimum point is distinguished from the maximum by the algebraic sign of the second derivative $F''(x)$. Formally, we have

Necessary Conditions:

$$F'(x^*) = 0. \tag{1.21}$$
$$F''(x^*) \geq 0; \tag{1.22}$$

Sufficient Conditions:

$$F'(x^*) = 0, \tag{1.23}$$
$$F''(x^*) > 0. \tag{1.24}$$

Note that the sufficient conditions require that $F''(x^*) > 0$, defining a *strong local minimizer* in contrast to a *weak local minimizer*, which may have $F''(x^*) = 0$. It is also important to observe that these conditions define a *local* rather than a *global* minimizer.

1.5 Newton's Method in Several Variables

The preceding sections have addressed problems involving a single variable. In this section, let us consider generalizing the discussion to functions of many variables. In particular, let us consider how to find the n-vector $\mathbf{x}^T = (x_1, \ldots, x_n)$ such that

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} c_1(\mathbf{x}) \\ \vdots \\ c_m(\mathbf{x}) \end{bmatrix} = \mathbf{0}. \tag{1.25}$$

For the present, let us assume that the number of constraints and variables is the same, i.e., $m = n$. Just as in one variable, a linear approximation to the constraint functions analogous to (1.5) is given by

$$\mathbf{c}(\bar{\mathbf{x}}) = \mathbf{c}(\mathbf{x}) + \mathbf{G}(\bar{\mathbf{x}} - \mathbf{x}). \tag{1.26}$$

where the *Jacobian matrix* $\mathbf{G}$ is defined by

$$\mathbf{G} \equiv \frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_1}{\partial x_2} & \cdots & \frac{\partial c_1}{\partial x_n} \\ \frac{\partial c_2}{\partial x_1} & \frac{\partial c_2}{\partial x_2} & \cdots & \frac{\partial c_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial c_m}{\partial x_1} & \frac{\partial c_m}{\partial x_2} & \cdots & \frac{\partial c_m}{\partial x_n} \end{bmatrix}. \tag{1.27}$$

By convention, the m rows of $\mathbf{G}$ correspond to constraints and the n columns to variables. As in one variable, if we require that $\mathbf{c}(\bar{\mathbf{x}}) = \mathbf{0}$ in (1.26), we can solve the linear system

$$\mathbf{G}\mathbf{p} = -\mathbf{c} \tag{1.28}$$

for the *search direction* $\mathbf{p}$, which leads to an iteration of the form

$$\bar{\mathbf{x}} = \mathbf{x} + \mathbf{p}. \tag{1.29}$$

Thus, each Newton iteration requires a linear approximation to the nonlinear constraints $\mathbf{c}$, followed by a step from $\mathbf{x}$ to the solution of the linearized constraints at $\bar{\mathbf{x}}$.

Figure 1.4 illustrates a typical situation when $n = m = 2$. It is important to remark that the multidimensional version of Newton's method shares all of the properties of its one-dimensional counterpart. Specifically, the method is quadratically convergent provided it is within a region of convergence, and it may diverge unless appropriate globalization strategies are employed. Furthermore, in order to solve (1.28) it is necessary that the Jacobian $\mathbf{G}$ be *nonsingular*, which is analogous to requiring that $c'(x) \neq 0$ in the univariate case. And, finally, it is necessary to actually compute $\mathbf{G}$, which can be costly.

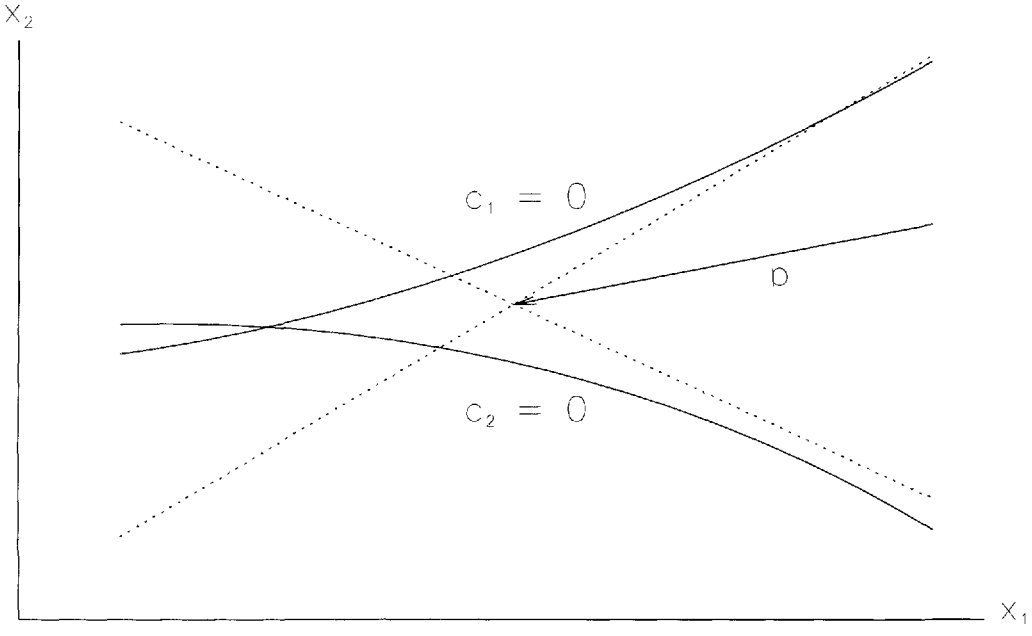


Figure 1.4: Newton's method in two variables.

1.6 Unconstrained Optimization

Let us now consider the multidimensional unconstrained minimization problem. Suppose we want to find the n-vector $\mathbf{x}^\mathsf{T} = (x_1, \ldots, x_n)$ such that the function $F(\mathbf{x}) = F(x_1, \ldots, x_n)$ is a minimum. Just as in the univariate case (1.18), let us approximate $F(\mathbf{x})$ by the first three terms in a Taylor series expansion about the point $\mathbf{x}$:

$$F(\overline{\mathbf{x}}) = F(\mathbf{x}) + \mathbf{g}^\mathsf{T}(\mathbf{x})(\overline{\mathbf{x}} - \mathbf{x}) + \frac{1}{2}(\overline{\mathbf{x}} - \mathbf{x})^\mathsf{T}\mathbf{H}(\mathbf{x})(\overline{\mathbf{x}} - \mathbf{x}). \tag{1.30}$$

The Taylor series expansion involves the n-dimensional *gradient* vector

$$\mathbf{g}(\mathbf{x}) \equiv \nabla_x F = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \vdots \\ \frac{\partial F}{\partial x_n} \end{bmatrix} \tag{1.31}$$

and the symmetric $n \times n$ *Hessian matrix*

$$\mathbf{H} \equiv \nabla_{xx}^2 F = \begin{bmatrix} \frac{\partial^2 F}{\partial x_1^2} & \frac{\partial^2 F}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_1 \partial x_n} \\ \frac{\partial^2 F}{\partial x_2 \partial x_1} & \frac{\partial^2 F}{\partial x_2^2} & \cdots & \frac{\partial^2 F}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 F}{\partial x_n \partial x_1} & \frac{\partial^2 F}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 F}{\partial x_n^2} \end{bmatrix} \tag{1.32}$$

It is common to define the *search direction* $\mathbf{p} = \bar{\mathbf{x}} - \mathbf{x}$ and then rewrite (1.30) as

$$F(\bar{\mathbf{x}}) = F(\mathbf{x}) + \mathbf{g}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p}. \tag{1.33}$$

The scalar term $\mathbf{g}^T\mathbf{p}$ is referred to as the *directional derivative* along $\mathbf{p}$ and the scalar term $\mathbf{p}^T\mathbf{H}\mathbf{p}$ is called the *curvature* or *second directional derivative* in the direction $\mathbf{p}$.

It is instructive to examine the behavior of the series (1.33). First, let us suppose that the expansion is about the minimum point $\mathbf{x}^*$. Now if $\mathbf{x}^*$ is a local minimum, then the objective function must be larger at all neighboring points, that is, $F(\bar{\mathbf{x}}) > F(\mathbf{x}^*)$. In order for this to be true, the slope in all directions must be zero, that is, $(\mathbf{g}^*)^T\mathbf{p} = \mathbf{0}$. which implies we must have

$$\mathbf{g}(\mathbf{x}^*) = \begin{bmatrix} g_1(\mathbf{x}^*) \\ \vdots \\ g_n(\mathbf{x}^*) \end{bmatrix} = \mathbf{0}. \tag{1.34}$$

This is just the multidimensional analogue of the condition (1.21). Furthermore, if the function curves up in all directions, the point $\mathbf{x}^*$ is called a strong local minimum and the third term in the expansion (1.33) must be positive:

$$\mathbf{p}^T\mathbf{H}^*\mathbf{p} > 0. \tag{1.35}$$

A matrix[1] that satisfies this condition is said to be *positive definite*. If there are some directions with zero curvature, i.e., $\mathbf{p}^T\mathbf{H}^*\mathbf{p} \geq 0$, then $\mathbf{H}^*$ is said to be *positive semidefinite*. If there are directions with both positive and negative curvature, the matrix is called *indefinite*. In summary,

Necessary Conditions:

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{1.36}$$
$$\mathbf{p}^T\mathbf{H}^*\mathbf{p} \geq 0; \tag{1.37}$$

Sufficient Conditions:

$$\mathbf{g}(\mathbf{x}^*) = \mathbf{0}, \tag{1.38}$$
$$\mathbf{p}^T\mathbf{H}^*\mathbf{p} > 0. \tag{1.39}$$

The preceding discussion was motivated by an examination of the Taylor series about the minimum point $\mathbf{x}^*$. Let us now consider the same quadratic model about an arbitrary

[1] $\mathbf{H}^* = \mathbf{H}(\mathbf{x}^*)$ (not the conjugate transpose, as in some texts).

point $\mathbf{x}$. Then it makes sense to choose a new point $\overline{\mathbf{x}}$ such that the gradient at $\overline{\mathbf{x}}$ is zero. The resulting linear approximation to the gradient is just

$$\overline{\mathbf{g}} = \mathbf{0} = \mathbf{g} + \mathbf{H}\mathbf{p}, \tag{1.40}$$

which can be solved to yield the Newton search direction

$$\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}. \tag{1.41}$$

Just as before, the Newton iteration is defined by (1.29). Since this iteration is based on finding a zero of the gradient vector, there is no guarantee that the step will move toward a local minimum rather than a stationary point or maximum. To preclude this, we must insist that the step be downhill, which requires satisfying the so-called descent condition

$$\mathbf{g}^T\mathbf{p} < 0. \tag{1.42}$$

It is interesting to note that, if we use the Newton direction (1.41), the descent condition becomes

$$\mathbf{g}^T\mathbf{p} = -\mathbf{g}^T\mathbf{H}^{-1}\mathbf{g} < 0, \tag{1.43}$$

which can only be true if the Hessian is positive definite, i.e., (1.35) holds.

1.7 Recursive Updates

Regardless of whether Newton's method is used for solving nonlinear equations, as in Section 1.5, or for optimization, as described in Section 1.6, it is necessary to compute derivative information. In particular, one must compute either the Jacobian matrix (1.27) or the Hessian matrix (1.32). For many applications, this can be a costly computational burden. Quasi-Newton methods attempt to construct this information recursively. A brief overview of the most important recursive updates is included, although a more complete discussion can be found in [46], [63], and [53].

The basic idea of a recursive update is to construct a new estimate of the Jacobian or Hessian using information from previous iterates. Most well-known recursive updates are of the form

$$\overline{\mathbf{B}} = \mathbf{B} + \mathcal{R}(\Delta\mathbf{c}, \Delta\mathbf{x}), \tag{1.44}$$

where the new estimate $\overline{\mathbf{B}}$ is computed from the old estimate $\mathbf{B}$. Typically, this calculation involves a low-rank modification $\mathcal{R}(\Delta\mathbf{c}, \Delta\mathbf{x})$ that can be computed from the previous step:

$$\Delta\mathbf{c} = \mathbf{c}^k - \mathbf{c}^{k-1}, \tag{1.45}$$

$$\Delta\mathbf{x} = \mathbf{x}^k - \mathbf{x}^{k-1}. \tag{1.46}$$

The usual way to construct the update is to insist that the secant condition

$$\overline{\mathbf{B}}\Delta\mathbf{x} = \Delta\mathbf{c} \tag{1.47}$$

hold and then construct an approximation $\overline{\mathbf{B}}$ that is "close" to the previous estimate $\mathbf{B}$. In Section 1.3, the simplest form of this condition (1.15) led to the secant method. In fact, the generalization of this formula, proposed in 1965 by C. G. Broyden [32], is

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{c} - \mathbf{B}\Delta\mathbf{x})\,(\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}, \tag{1.48}$$

which is referred to as the *secant* or *Broyden update*. The recursive formula constructs a rank-one modification that satisfies the secant condition and minimizes the Frobenius norm between the estimates.

When a quasi-Newton method is used to approximate the Hessian matrix, as required for minimization, one cannot simply replace $\Delta\mathbf{c}$ with $\Delta\mathbf{g}$ in the secant update. In particular, the matrix $\overline{\mathbf{B}}$ constructed using (1.48) is not symmetric. However, there is a rank-one update that does maintain symmetry, known as the *symmetric rank-one* (SR1) update:

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}, \tag{1.49}$$

where $\Delta\mathbf{g} \equiv \mathbf{g}^k - \mathbf{g}^{k-1}$. While the SR1 update does preserve symmetry, it does not necessarily maintain a positive definite approximation. In contrast, the update

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{\Delta\mathbf{g}(\Delta\mathbf{g})^{\mathsf{T}}}{(\Delta\mathbf{g})^{\mathsf{T}}\Delta\mathbf{x}} - \frac{\mathbf{B}\Delta\mathbf{x}(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{B}}{(\Delta\mathbf{x})^{\mathsf{T}}\mathbf{B}\Delta\mathbf{x}} \tag{1.50}$$

is a rank-two positive definite secant update provided $(\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{g} > 0$ is enforced at each iteration. This update was discovered independently by Broyden [33], Fletcher [52], Goldfarb [65], and Shanno [98] in 1970 and is known as the *BFGS update*.

The effective computational implementation of a quasi-Newton update introduces a number of additional considerations. When solving nonlinear equations, the search direction from (1.28) is $\mathbf{p} = -\mathbf{G}^{-1}\mathbf{c}$, and for optimization problems the search direction given by (1.41) is $\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$. Since the search direction calculation involves the matrix inverse (either $\mathbf{G}^{-1}$ or $\mathbf{H}^{-1}$), one apparent simplification is to apply the recursive update directly to the inverse. In this case, the search direction can be computed simply by computing the matrix-vector product. This approach was proposed by Broyden for nonlinear equations, but has been considerably less successful in practice than the update given by (1.48), and is known as "Broyden's bad update." For unconstrained minimization, let us make the substitutions $\Delta\mathbf{x} \rightarrow \Delta\mathbf{g}$, $\Delta\mathbf{g} \rightarrow \Delta\mathbf{x}$, and $\mathbf{B} \rightarrow \mathbf{B}^{-1}$ in (1.50). By computing the inverse of the resulting expression, one obtains

$$\overline{\mathbf{B}} = \mathbf{B} + \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})(\Delta\mathbf{g})^{\mathsf{T}} + \Delta\mathbf{g}(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}}{(\Delta\mathbf{g})^{\mathsf{T}}\Delta\mathbf{x}} - \sigma\Delta\mathbf{g}(\Delta\mathbf{g})^{\mathsf{T}}, \tag{1.51}$$

where

$$\sigma = \frac{(\Delta\mathbf{g} - \mathbf{B}\Delta\mathbf{x})^{\mathsf{T}}\Delta\mathbf{x}}{(\Delta\mathbf{g}^{\mathsf{T}}\Delta\mathbf{x})^2}.$$

This so-called inverse positive definite secant update is referred to as the *DFP update* for its discoverers Davidon [43] and Fletcher and Powell [55].

Even though many recursive updates can be applied directly to the inverse matrices, most practical implementations do not use this approach. When the matrices $\mathbf{G}$ and/or $\mathbf{H}$ are singular, the inverse matrices do not exist. Consequently, it is usually preferable to work directly with $\mathbf{G}$ and $\mathbf{H}$. There are at least three issues that must be addressed by an effective implementation, namely efficiency, numerical conditioning, and storage. The solution of a dense linear system, such as (1.28) or (1.40), requires $\mathcal{O}(n^3)$ operations, compared with the $\mathcal{O}(n^2)$ operations needed to compute the matrix-vector product (1.41). However, this penalty can be avoided by implementing the update in "factored form." For example, the (positive definite) Hessian matrix can be written in terms of its Cholesky factorization as $\mathbf{H} = \mathbf{R}\mathbf{R}^{\mathsf{T}}$. Since the recursive update formulas represent low-rank modifications to $\mathbf{H}$, it is possible to derive low-rank modifications to the factors $\mathbf{R}$. By updating the matrices in factored form, the cost of computing the search direction can be reduced to $\mathcal{O}(n^2)$ operations, just as when the inverse is recurred directly. Furthermore, when the matrix factorizations are available, it is also possible to deal with rank deficiencies in a more direct fashion. Finally, when storage is an issue, the matrix at iteration k can be represented as the sum of L quasi-Newton updates to the original estimate $\mathbf{B}_0$ in the form

$$\mathbf{B}_k = \mathbf{B}_0 + \sum_{i=1}^{L} \mathbf{u}_i \mathbf{u}_i^{\mathsf{T}} - \sum_{i=1}^{L} \mathbf{v}_i \mathbf{v}_i^{\mathsf{T}}, \tag{1.52}$$

where the vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ denote information from iteration i. If the initial estimate $\mathbf{B}_0$ requires relatively little storage (e.g., is diagonal), then all operations involving the matrix $\mathbf{B}_k$ at iteration k can be performed without explicitly forming the (dense) matrix $\mathbf{B}_k$. This technique, called a *limited memory update*, only requires storing the vectors $\mathbf{u}$ and $\mathbf{v}$ over the previous L iterations.

We have motivated the use of a recursive update as a way to construct Jacobian and/or Hessian information. However, we have not discussed how fast an iterative sequence will converge when the recursive update is used instead of the exact information. All of the methods that use a recursive update exhibit *superlinear* convergence provided the matrices are nonsingular. In general, superlinear convergence is not as fast as quadratic convergence. One way to measure the rate of convergence is to compare the behavior of a Newton method and a quasi-Newton method on a quadratic function of n variables. Newton's method will terminate in one step, assuming finite-precision arithmetic errors are negligible. In contrast, a quasi-Newton method will terminate in at most n steps, provided the steplength α is chosen at each iteration to minimize the value of the objective function at the new point $F(\bar{\mathbf{x}}) = F(\mathbf{x} + \alpha\mathbf{p})$. The process of adjusting α is called a *line search* and will be discussed in Section 1.11.

1.8 Equality-Constrained Optimization

The preceding sections describe how Newton's method can be applied either to optimize an objective function $F(\mathbf{x})$ *or* to satisfy a set of constraints $\mathbf{c}(\mathbf{x}) = \mathbf{0}$. Suppose now that we want to do both, that is, choose the variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{1.53}$$

subject to the $m \leq n$ constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{0}. \tag{1.54}$$

The classical approach is to define the *Lagrangian*

$$L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}), \tag{1.55}$$

where $\boldsymbol{\lambda}$ is an m-vector of *Lagrange multipliers*.

In a manner analogous to the unconstrained case, optimality requires that derivatives with respect to both $\mathbf{x}$ and $\boldsymbol{\lambda}$ be zero. More precisely, necessary conditions for the point $(\mathbf{x}^*, \boldsymbol{\lambda}^*)$ to be an optimum are

$$\nabla_x L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}, \tag{1.56}$$

$$\nabla_\lambda L(\mathbf{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}. \tag{1.57}$$

The gradient of L with respect to $\mathbf{x}$ is

$$\nabla_x L = \mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} = \nabla F - \sum_{i=1}^{m} \lambda_i \nabla c_i \tag{1.58}$$

and the gradient of L with respect to $\boldsymbol{\lambda}$ is

$$\nabla_\lambda L = -\mathbf{c}(\mathbf{x}). \tag{1.59}$$

Just as in the unconstrained case, these conditions do not distinguish between a point that is a minimum, a maximum, or simply a stationary point. As before, we require conditions on the curvature of the objective. Let us define the *Hessian of the Lagrangian* to be

$$\mathbf{H}_L = \nabla^2_{xx} L = \nabla^2_{xx} F - \sum_{i=1}^{m} \lambda_i \nabla^2_{xx} c_i. \tag{1.60}$$

Then a sufficient condition is that

$$\mathbf{v}^{\mathsf{T}}\mathbf{H}_L\mathbf{v} > \mathbf{0} \tag{1.61}$$

for any vector $\mathbf{v}$ in the constraint tangent space. If one compares (1.35) with (1.61), an important difference emerges. For the unconstrained case, we require that the curvature be positive in *all* directions $\mathbf{p}$. However, (1.61) applies only to directions $\mathbf{v}$ in the constraint tangent space.

Example 1.2. To fully appreciate the meaning of these conditions, consider the simple example problem with two variables and one constraint illustrated in Figure 1.5. Let us minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraint

$$c(\mathbf{x}) = x_1 + x_2 - 2 = 0.$$

The solution is at $\mathbf{x}^* = (1, 1)$. Now for this example, the Jacobian is just $\mathbf{G} = \nabla c^{\mathsf{T}} = (1, 1)$, which is a vector orthogonal to the constraint. Consequently, if we choose $\mathbf{v}^{\mathsf{T}} = (-a, a)$ for some constant $a \neq 0$, the vector $\mathbf{v}$ is tangent to the constraint, which

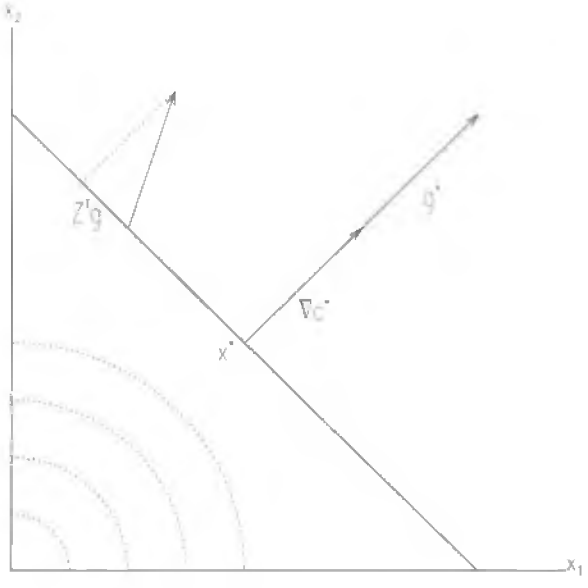


Figure 1.5: Equality-constrained example.

can be readily verified since $\mathbf{Gv} = (1)(-a) + (1)(a) = 0$. At $\mathbf{x}^*$, the gradient is a linear combination of the constraint gradients, i.e., (1.58) becomes

$$\mathbf{g} - \mathbf{G}^T\boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix} 2 = \mathbf{0}.$$

Furthermore, from (1.61), the curvature

$$\mathbf{v}^T\mathbf{H}_L\mathbf{v} = [-a, a] \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -a \\ a \end{bmatrix} = 4a^2$$

is clearly positive.

Notice that, at the optimal solution, the gradient vector is orthogonal to the constraint surface, or equivalently that the projection of the gradient vector onto the constraint surface is zero. A second point is also illustrated in Figure 1.5, demonstrating that the projection of the gradient is nonzero at the suboptimal point. Apparently then, there is a matrix $\mathbf{Z}$ that projects the gradient onto the constraint surface. This implies that an equivalent form of the necessary condition (1.56) is to require that the *projected gradient* be zero, i.e.,

$$\mathbf{Z}^T\mathbf{g} = \mathbf{0}. \tag{1.62}$$

In a similar fashion, one can define an equivalent form of the sufficient condition (1.61) in terms of the *projected Hessian* matrix

$$\mathbf{Z}^T\mathbf{H}_L\mathbf{Z}. \tag{1.63}$$

In other words, we require the projected Hessian matrix (1.63) to be positive definite. Notice that the projection matrix $\mathbf{Z}$ has n rows and $n_d = n - m$ columns. We refer to

the quantity n_d as the *number of degrees of freedom.* Observe also that the projected gradient (1.62) is a vector of length n_d and the projected Hessian (1.63) is $n_d \times n_d$. In general, the choice of $\mathbf{Z}$ is not unique, although for the example problem one can choose any scalar multiple of $\mathbf{Z}^T = (-1, 1)$.

1.8.1 Newton's Method

Let us now apply Newton's method to find the values of $(\mathbf{x}, \boldsymbol{\lambda})$ such that the necessary conditions (1.56) and (1.57) are satisfied. Proceeding formally to construct a Taylor series expansion analogous to (1.26), the expansion about $(\mathbf{x}, \boldsymbol{\lambda})$ for the functions (1.58) and (1.59) is just

$$\mathbf{0} = \mathbf{g} - \mathbf{G}^T\boldsymbol{\lambda} + \mathbf{H}_L(\bar{\mathbf{x}} - \mathbf{x}) - \mathbf{G}^T(\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}), \tag{1.64}$$

$$\mathbf{0} = -\mathbf{c} - \mathbf{G}(\bar{\mathbf{x}} - \mathbf{x}). \tag{1.65}$$

After simplification, these equations lead to the linear system analogous to that given by (1.28), which is called the Kuhn–Tucker (KT) or Karush–Kuhn–Tucker (KKT) system:

$$\begin{bmatrix} \mathbf{H}_L & \mathbf{G}^T \\ \mathbf{G} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \bar{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{c} \end{bmatrix}, \tag{1.66}$$

where $\mathbf{p}$ is the search direction for a step $\bar{\mathbf{x}} = \mathbf{x} + \mathbf{p}$ and $\bar{\boldsymbol{\lambda}}$ is the vector of Lagrange multipliers at the new point. Notice that the system is written in terms of the change in the variables, i.e., $\mathbf{p} = \bar{\mathbf{x}} - \mathbf{x}$, but does not involve the change in the multipliers, i.e., $\bar{\boldsymbol{\lambda}} - \boldsymbol{\lambda}$. Instead, it is preferable to explicitly eliminate the term $\mathbf{G}^T\boldsymbol{\lambda}$, which appears in the Taylor series approximation (1.64). Thus, in this instance, Newton's method is based on a linear approximation to the constraints *and* a linear approximation to the gradients. As in the unconstrained optimization case, a linear approximation to the gradient is equivalent to making a quadratic model for the quantity being optimized. It is important to note that the quadratic approximation is made to the Lagrangian (1.55) and not just the objective function F. Because of the underlying quadratic-linear model, Newton's method will converge in one step for a quadratic objective with linear equality constraints.

Although the derivation of the KKT system (1.66) was motivated by a Taylor series expansion, an alternative motivation is to choose $\mathbf{p}$ to minimize the quadratic objective

$$\mathbf{g}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} \tag{1.67}$$

subject to the linear constraints

$$\mathbf{G}\mathbf{p} = -\mathbf{c}. \tag{1.68}$$

It is straightforward to demonstrate that the optimality conditions for this quadratic-linear optimization subproblem are given by the KKT system (1.66).

1.9 Inequality-Constrained Optimization

Section 1.8 introduced the equality-constrained optimization problem. In this section, we consider a problem with inequality constraints. Suppose that we want to choose the variables $\mathbf{x}$ to minimize

$$F(\mathbf{x}) \tag{1.69}$$

subject to the m inequality constraints

$$\mathbf{c}(\mathbf{x}) \geq \mathbf{0}. \tag{1.70}$$

In contrast to equality-constrained problems, which require $m \leq n$, the number of inequality constraints can exceed the number of variables. A point that satisfies the constraints is said to be *feasible* and the collection of all feasible points is called the *feasible region*. Conversely, points in the infeasible region violate one or more of the constraints.

Inequality-constrained problems are characterized by a fundamental concept. Specifically, at the solution $\mathbf{x}^*$,

1. some of the constraints will be satisfied as equalities, that is,

$$c_i(\mathbf{x}^*) = 0 \qquad \text{for} \qquad i \in \mathcal{A}, \tag{1.71}$$

   where $\mathcal{A}$ is called the *active set*, and

2. some constraints will be strictly satisfied, that is,

$$c_i(\mathbf{x}^*) > 0 \qquad \text{for} \qquad i \in \mathcal{A}', \tag{1.72}$$

   where $\mathcal{A}'$ is called the *inactive set*.

Thus, the total set of constraints is partitioned into two subsets, namely the active and inactive constraints. Clearly, the active constraints can be treated using the methods described in Section 1.8. Obviously, an inequality-constrained optimization algorithm needs some mechanism to identify the active constraints. This mechanism is referred to as an *active set strategy*. Fortunately, there is an additional necessary condition for inequality-constrained problems that is essential to active set strategies. Specifically, at the solution, the algebraic sign of the Lagrange multipliers must be correct, that is, we must have

$$\lambda_i^* \geq 0 \qquad \text{for} \qquad i \in \mathcal{A}. \tag{1.73}$$

To illustrate the impact of inequality constraints, let us consider two examples that are modifications of Example 1.2 presented in Section 1.8.

Example 1.3. The left side of Figure 1.6 illustrates the following problem: Minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraint

$$c(\mathbf{x}) = 2 - x_1 - x_2 \geq 0.$$

The gradient and Jacobian are given by

$$\mathbf{g} = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix}, \qquad \mathbf{G} = \begin{bmatrix} -1 & -1 \end{bmatrix}.$$

Now at the point $\mathbf{x}^\mathsf{T} = (1, 1)$, the optimality conditions are

$$\mathbf{0} = \mathbf{g} - \mathbf{G}^\mathsf{T}\boldsymbol{\lambda} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} -1 \\ -1 \end{bmatrix} [-2]$$

and since $\lambda = -2 < 0$, the constraint should be deleted from the active set. The solution is $\mathbf{x}^\mathsf{T} = (0, 0)$.

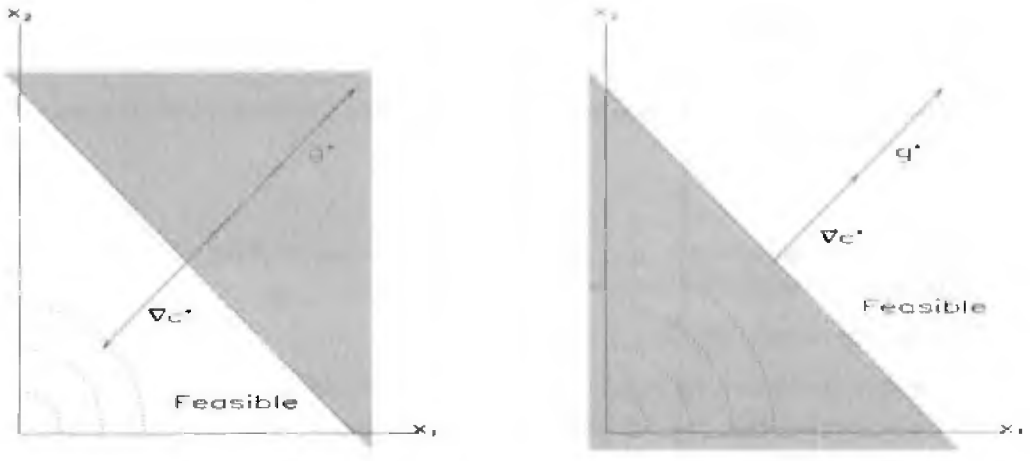


Figure 1.6: Inequality-constrained examples.

Example 1.4. On the other hand, if the sense of the inequality is reversed, we must minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the inequality

$$c(\mathbf{x}) = x_1 + x_2 - 2 \geq 0.$$

This problem is illustrated on the right side of Figure 1.6. At the point $\mathbf{x}^\mathsf{T} = (1, 1)$, the Lagrange multiplier $\lambda = 2 > 0$. Consequently, the constraint is *active* and cannot be deleted. The solution is $\mathbf{x}^\mathsf{T} = (1, 1)$.

1.10 Quadratic Programming

An important special form of optimization problem is referred to as the quadratic programming (QP) problem. A QP problem is characterized by

- a quadratic objective

$$F(\mathbf{x}) = \mathbf{g}^\mathsf{T}\mathbf{x} + \frac{1}{2}\mathbf{x}^\mathsf{T}\mathbf{H}\mathbf{x} \tag{1.74}$$

- and linear constraints

$$\mathbf{A}\mathbf{x} = \mathbf{a}, \tag{1.75}$$
$$\mathbf{B}\mathbf{x} \geq \mathbf{b}, \tag{1.76}$$

where $\mathbf{H}$ is the positive definite Hessian matrix.

Let us now outline an active set method for solving this QP problem using the concepts introduced in Sections 1.8 and 1.9. Assume that an estimate of the active set $\mathcal{A}^0$ is given in addition to a feasible point $\mathbf{x}^0$. A quadratic programming algorithm proceeds as follows:

1. Compute the minimum of the quadratic objective subject to the constraints in the active set estimate $\mathcal{A}$ treated as *equalities*, i.e., solve the KKT system (1.66).

2. Take the largest possible step in the direction $\mathbf{p}$ that does not violate any *inactive* inequalities, that is,

$$\overline{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{p}, \tag{1.77}$$

   where the steplength $0 \leq \alpha \leq 1$ is chosen to maintain feasibility with respect to the inactive inequality constraints.

3. If the step is restricted, i.e., $\alpha < 1$, then

   - *add* the limiting inequality to the active set $\mathcal{A}$ and return to step 1.
   - otherwise, take the full step ($\alpha = 1$) and check the sign of the Lagrange multipliers and
     - if all of the inequalities have positive multipliers, terminate the algorithm;
     - otherwise, *delete* the inequality with the most negative λ from the active set and return to step 1.

Notice that step 1 requires the solution of the KKT system (1.66) corresponding to the set of active constraints defined by $\mathcal{A}$. In particular, (1.66) becomes

$$\begin{bmatrix} \mathbf{H} & \mathbf{A}^T & \widetilde{\mathbf{B}}^T \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \widetilde{\mathbf{B}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{p} \\ \overline{\boldsymbol{\eta}} \\ \overline{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \mathbf{g} \\ \mathbf{a} \\ \widetilde{\mathbf{b}} \end{bmatrix}, \tag{1.78}$$

where the vector $\widetilde{\mathbf{b}}$ denotes the subset of the vector $\mathbf{b}$ corresponding to the *active* inequality constraints and $\widetilde{\mathbf{B}}$ is the corresponding Jacobian matrix for the constraints in $\mathcal{A}$. The Lagrange multipliers corresponding to the equality constraints are denoted by $\boldsymbol{\eta}$. For the purposes of this discussion, it suffices to say that any reasonable method for solving (1.78) can be used. However, in practice it is extremely important that this solution be computed in an efficient and numerically stable way. This is especially true because every time the active set changes in step 3, it is necessary to solve a different KKT system to compute the new step. This is because each time the active set changes, the matrix $\widetilde{\mathbf{B}}$ is altered by adding or deleting a row. Most efficient QP implementations compute the new step by modifying the previously computed solution and the associated matrix factorizations. In general, the computational expense of the QP problem is dominated by the linear algebra and one should *not* use a "brute force" solution to the KKT system! While space precludes a more detailed discussion of this subject, the interested reader should consult the book by Gill, Murray, and Wright [64]. An approach that is particularly efficient for large, sparse applications will be described in Section 2.3.

It should also be apparent that the number of QP iterations is determined by the initial active set $\mathcal{A}^0$. If the initial active set is correct, i.e., $\mathcal{A}^0 = \mathcal{A}^*$, then the QP algorithm will compute the solution in one iteration. Conversely, many QP iterations may be required if the initial active set differs substantially from the final one. Additional complications can arise when the Hessian matrix $\mathbf{H}$ is indefinite because the possibility of an unbounded solution exists. Furthermore, when a multiplier $\lambda_k \approx 0$, considerable care must be exercised when deleting a constraint in step 3.

As stated, the QP algorithm requires a feasible initial point $\mathbf{x}^0$, which may require a special "phase-1" procedure in the software. It is also worth noting that when $\mathbf{H} = \mathbf{0}$, the objective is linear and the optimization problem is referred to as an LP problem. In this case, the active set algorithm described is equivalent to the *simplex method* proposed by Dantzig [42] in 1947. However, it is important to note that a *unique* solution to a linear program will have n active constraints, whereas there may be many degrees of freedom at the solution of a quadratic program.

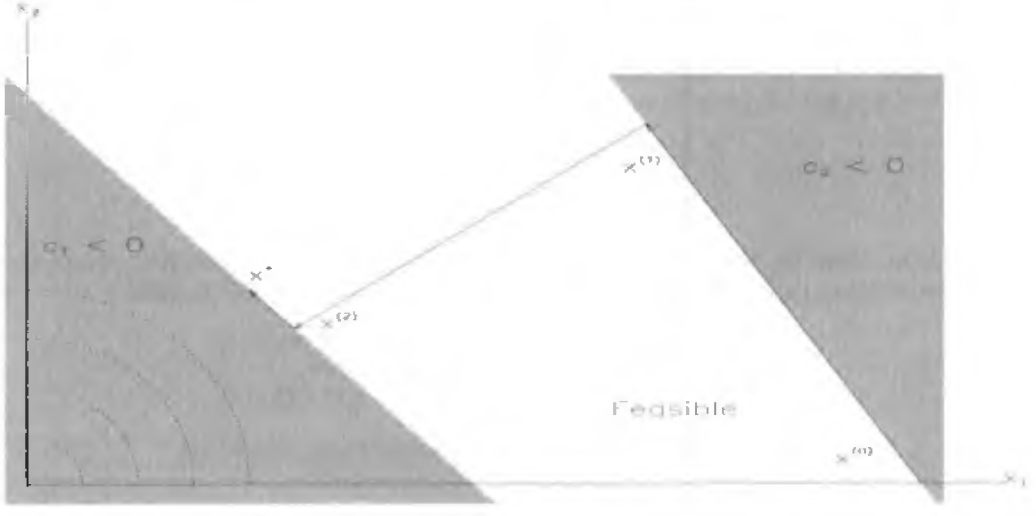


Figure 1.7: QP example.

Iteration	x_1	x_2	Active Set $\mathcal{A}$
0	4	0	$\mathcal{A}^0 = \{c_2\}$
1	2.76923	1.84615	Delete c_2 ($\lambda_2 = -5.53846$).
2	1.2	0.8	Add c_1 ($\alpha = 0.566667$).
3	1	1	$\mathcal{A}^* = \{c_1\}$

Table 1.2: QP iterations.

Example 1.5. To illustrate how a QP algorithm works, let us consider minimizing

$$F(\mathbf{x}) = x_1^2 + x_2^2$$

subject to the constraints

$$c_1(\mathbf{x}) = x_1 + x_2 - 2 \geq 0,$$
$$c_2(\mathbf{x}) = 4 - x_1 - \frac{2}{3}x_2 \geq 0.$$

This problem is illustrated in Figure 1.7. Table 1.2 summarizes the QP steps assuming the iteration begins at $\mathbf{x}^T = (4, 0)$ with $\mathcal{A}^0 = \{c_2\}$. The first step requires solving the

KKT system (1.78) evaluated at the point $\mathbf{x}^T = (4, 0)$, that is,

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -\frac{2}{3} \\ -1 & -\frac{2}{3} & 0 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \\ \bar{\lambda}_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 0 \\ 0 \end{bmatrix}.$$

After this step, the new point is at $\bar{x}_1 = x_1 + p_1 = 2.76923$, and $\bar{x}_2 = x_2 + p_2 = 1.84615$. Since the Lagrange multiplier is negative at the new point, constraint c_2 can be deleted from the active set.

The second QP step begins at $\mathbf{x}^T = (2.76923, 1.84615)$ with no constraints active, i.e., $\mathcal{A}^1 = \{\emptyset\}$. The new step is computed from the KKT system

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \end{bmatrix} = \begin{bmatrix} 5.53846 \\ 3.69230 \end{bmatrix}.$$

However, in this case, it is not possible to take a full step because the first constraint would be violated. Instead, one finds $\bar{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{p}$ with $\alpha = 0.566667$, to give

$$\bar{\mathbf{x}} = \begin{bmatrix} 1.2 \\ 0.8 \end{bmatrix} = \begin{bmatrix} 2.76923 \\ 1.84615 \end{bmatrix} - 0.566667 \begin{bmatrix} 2.76923 \\ 1.84615 \end{bmatrix}.$$

In general, the length of the step is given by the simple linear prediction

$$\alpha = \min \left[\frac{-c_i(\mathbf{x})}{\mathbf{p}^T \nabla c_i} \right] > 0 \qquad \text{for} \qquad i \in \mathcal{A}'. \tag{1.79}$$

Essentially, one must compute the largest step that will not violate any of the (currently) inactive inequality constraints.

Since the second QP iteration terminates by encountering the first constraint c_1, it must be added to the active set so that $\mathcal{A}^2 = \{c_1\}$. Once again the new step is computed from the KKT system, which in this case is just

$$\begin{bmatrix} 2 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} -p_1 \\ -p_2 \\ \bar{\lambda}_1 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 1.6 \\ 0 \end{bmatrix}.$$

After computing the new point, the solution is given by $\mathbf{x}^* = (1, 1)$, and the final active set is $\mathcal{A}^* = \{c_1\}$, with optimal Lagrange multipliers $\boldsymbol{\lambda}^* = (2, 0)$.

1.11 Globalization Strategies

1.11.1 Merit Functions

To this point, the development has stressed the application of Newton's method. However, even for the simplest one-dimensional applications described in Section 1.2, Newton's method has deficiencies. Methods for correcting the difficulties with Newton's method are referred to as *globalization strategies*. There are two primary roles performed by a globalization strategy, namely,

1. detecting a problem with the (unmodified) Newton's method and
2. correcting the deficiency.

It should be emphasized that the goal of all globalization strategies is to do *nothing!* Clearly, near a solution it is desirable to retain the quadratic convergence of a Newton step and, consequently, it is imperative that modifications introduced by the globalization process not impede the ultimate behavior.

Referring to (1.28) and (1.29), all Newton iterations are based on linearizations of the form

$$\mathbf{G}\mathbf{p} = -\mathbf{c} \tag{1.80}$$

for the search direction $\mathbf{p}$, which leads to an iteration of the form

$$\overline{\mathbf{x}} = \mathbf{x} + \mathbf{p}. \tag{1.81}$$

If Newton's method is working properly, the sequence of iterates $\{\mathbf{x}^{(k)}\}$ should converge to the solution $\mathbf{x}^*$. In practice, of course, the solution $\mathbf{x}^*$ is unknown, and we must detect whether the sequence is converging properly using information that is available. The most common way to measure progress is to assign some *merit function*, M, to each iterate $\mathbf{x}^{(k)}$ and then insist that

$$M(\mathbf{x}^{(k+1)}) < M(\mathbf{x}^{(k)}). \tag{1.82}$$

This is one approach for detecting when the Newton sequence is working. When Newton's method is used for unconstrained optimization, an obvious merit function is just $M(\mathbf{x}) = F(\mathbf{x})$. When Newton's method is used to find the root of many functions, as in (1.80), assigning a single value to quantify "progress" is no longer quite so obvious. The most commonly used merit function for nonlinear equations is

$$M(\mathbf{x}) = \frac{1}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}). \tag{1.83}$$

If the only purpose for constructing a merit function is to quantify progress in the iterative sequence, then any other norm is also suitable, e.g., we could use

$$M(\mathbf{x}) = \|\mathbf{c}\|_1 = \sum_{i=1}^{m} |c_i|,$$

$$M(\mathbf{x}) = \|\mathbf{c}\|_2 = \left(\sum_{i=1}^{m} c_i^2\right)^{\frac{1}{2}},$$

or

$$M(\mathbf{x}) = \|\mathbf{c}\|_\infty = \max_{i=1}^{m} |c_i|.$$

Choosing a merit function for constrained optimization is still more problematic, for it is necessary to somehow balance the (often) conflicting goals of reducing the objective function while satisfying the constraints. To this point, we have motivated the discussion of a merit function as an artifice to "fix" Newton's method when it is not converging. An alternative approach is to construct some other function P, whose *unconstrained*

minimum is either the desired constrained solution $\mathbf{x}^*$ or is related to it in a known way. Thus, we might consider solving a sequence of unconstrained problems whose solutions approach the desired constrained optimum. This point of view is fundamental to so-called penalty function methods. One possible candidate is the quadratic penalty function

$$P(\mathbf{x}, \rho) = F(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}), \tag{1.84}$$

where ρ is called the *penalty weight* or *penalty parameter*. When this penalty function is minimized for successively larger values of the penalty weight ρ, it can be shown that the unconstrained minimizers approach the constrained solution. Unfortunately, from a computational viewpoint, this is unattractive since, as the parameter $\rho \to \infty$, the successive unconstrained optimization problems become harder and harder to solve. An alternative, which avoids this ill-conditioning, is the so-called *augmented Lagrangian function* formed by combining the Lagrangian (1.55) with a constraint penalty of the form (1.83) to yield

$$P(\mathbf{x}, \boldsymbol{\lambda}, \rho) = L(\mathbf{x}, \boldsymbol{\lambda}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) + \frac{\rho}{2}\mathbf{c}^{\mathsf{T}}(\mathbf{x})\mathbf{c}(\mathbf{x}). \tag{1.85}$$

It can be shown that the unconstrained minimum of this function is equal to the constrained minimum $\mathbf{x}^*$ for a *finite* value of the parameter ρ. Unfortunately, in practice, choosing a "good" value for the penalty weight is nontrivial. If ρ is too large, the unconstrained optimization problem is ill-conditioned and, consequently, very difficult to solve. If ρ is too small, the unconstrained minimum of the augmented Lagrangian may be unbounded and/or the problem may be ill-conditioned. On the other hand, some of these drawbacks are not as critical when the augmented Lagrangian is used only to measure progress as part of a globalization strategy. Thus, we are led to the notion of computing the Newton step $\mathbf{p}$ in the usual way and then measuring progress by insisting that

$$M(\mathbf{x}^{(k+1)}, \boldsymbol{\lambda}^{(k+1)}, \rho) < M(\mathbf{x}^{(k)}, \boldsymbol{\lambda}^{(k)}, \rho), \tag{1.86}$$

where the merit function $M(\mathbf{x}, \boldsymbol{\lambda}, \rho) = P(\mathbf{x}, \boldsymbol{\lambda}, \rho)$. In summary, a merit function can be used to quantitatively decide whether or not Newton's method is working.

1.11.2 Line-Search Methods

Assuming for the moment that a merit function is used as an indicator of progress, how does one alter Newton's method when necessary? One approach is to alter the *magnitude* of the step using a *line-search* method. A second approach is to change both the magnitude and the *direction* using a *trust-region* method. The basic notion of a line-search method is to replace the Newton step (1.81) with

$$\overline{\mathbf{x}} = \mathbf{x} + \alpha\mathbf{p}. \tag{1.87}$$

Using this expression, it then follows that the merit function can be written as a function of the single variable α:

$$M(\overline{\mathbf{x}}) = M(\mathbf{x} + \alpha\mathbf{p}) = M(\alpha). \tag{1.88}$$

Furthermore, it is reasonable to choose the steplength α such that $M(\alpha)$ is approximately minimized. Most modern line-search implementations begin with the full Newton step,

i.e., $\alpha^{(0)} = 1$. Then estimates are computed until a steplength $\alpha^{(k)}$ is found that satisfies a *sufficient decrease* condition given by the *Goldstein-Armijo* condition

$$0 < -\kappa_1 \alpha^{(k)} M'(0) \leq M(0) - M(\alpha^{(k)}) \leq -\kappa_2 \alpha^{(k)} M'(0). \tag{1.89}$$

$M'(0) = (\nabla M(0))^{\mathsf{T}} \mathbf{p}$ is the direction derivative at $\alpha = 0$. with the constants κ_1 and κ_2 satisfying $0 < \kappa_1 \leq \kappa_2 < 1$. Typical values for the constants are $\kappa_1 = 10^{-4}$ and $\kappa_2 = 0.9$. and do *not* require an "accurate" minimization of M. Nevertheless, most line-search implementations are quite sophisticated and use quadratic and/or cubic polynomial interpolation in conjunction with some type of safeguarding procedure. Furthermore, special-purpose line-search procedures are often employed for different merit functions, e.g., see Murray and Wright [85]. In fact, we have already described two applications that may be viewed as special-purpose line-search algorithms. First, if a quasi-Newton update is used to construct the Hessian, an "exact" line search will produce termination after n steps on a quadratic function. Second, when solving a quadratic program, the steplength given by (1.79) is actually the minimum of the quadratic function restricted such that inactive inequalities are not violated.

Example 1.6. To illustrate how Newton's method works when a line-search strategy is incorporated, let us consider the simple nonlinear system

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} 1 - x_1 \\ 10(x_2 - x_1^2) \end{bmatrix} = \mathbf{0}.$$

If the merit function $M(\mathbf{x})$ (1.83) is used to monitor progress and the steplength α is adjusted using a line search with polynomial interpolation, one obtains the iteration history summarized in Table 1.3. Notice that every iteration produces a reduction in the merit function as it must. However, in order to achieve this monotonic behavior, it is necessary to modify the steplength on nearly all iterations. Quadratic convergence is observed only during the final few iterations.

Iter.	x_1	x_2	α	$M(\mathbf{x})$
0	-1.2000	1.0000	$--$	12.100
1	-1.0743	0.72336	0.57157×10^{-1}	11.425
2	-0.94553	0.47352	0.62059×10^{-1}	10.734
3	-0.81340	0.25221	0.67917×10^{-1}	10.025
4	-0.67732	0.61558×10^{-1}	0.75040×10^{-1}	9.2952
5	-0.53662	-0.95719×10^{-1}	0.83883×10^{-1}	8.5411
6	-0.39041	-0.21613	0.95147×10^{-1}	7.7580
7	-0.23751	-0.29499	0.10997	6.9398
8	-0.76248×10^{-1}	-0.32580	0.13031	6.0775
9	0.66751×10^{-1}	0.30355	0.13287	5.1787
10	0.22101	-0.23204	0.16529	4.2483
11	0.36706	-0.11482	0.18748	3.3142
12	0.55206	0.93929×10^{-1}	0.29229	2.3230
13	1.0000	0.79935	1.0000	2.0131
14	1.0000	1.0000	1.0000	0.12658×10^{-18}

Table 1.3: Newton's method with line search.

Iter.	x_1	x_2	α	$M(\mathbf{x})$
0	-1.2000	1.0000	$--$	12.100
1	1.0000	-3.8400	1.0000	1171.3
2	1.0000	1.0000	1.0000	0.85927×10^{-14}

Table 1.4: Newton's method without line search.

It is interesting to compare the behavior of Newton's method, which has a line-search globalization procedure, to that of an unmodified Newton algorithm. Table 1.4 summarizes the iterations when Newton's method is applied *without* a line search. Notice that all steps have $\alpha = 1$. Furthermore, observe that the first step produced a large increase in the constraint error as measured by the merit function M. Nevertheless, in this case, the unmodified Newton method was significantly more efficient than the approach with a line search. For this example, the line search is not only unnecessary but also inefficient. This suggests two questions. First, do we need a globalization strategy? Clearly, the answer is yes. Second, is a line search with merit function the most efficient strategy? Perhaps not. The numerical results suggest that there is considerable room for improvement provided a mechanism can be devised that is sufficiently robust.

1.11.3 Trust-Region Methods

A line-search globalization strategy computes the search direction $\mathbf{p}$ and then adjusts the steplength parameter α in order to define the iteration (1.87). A second alternative is to adjust both the magnitude and direction of the vector $\mathbf{p}$. Treating the current point $\mathbf{x}$ as fixed, suppose that we want to choose the variables $\mathbf{p}$ to minimize

$$F(\mathbf{x}+\mathbf{p}) \approx F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{1.90}$$

subject to the constraint

$$\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p} \leq \delta^2. \tag{1.91}$$

We assume that we can *trust* the prediction $F(\mathbf{x}+\mathbf{p})$ as long as the points lie within the region defined by the *trust radius* δ. Proceeding formally to define the Lagrangian for this problem, one obtains

$$L_T(\mathbf{p},\tau) = F(\mathbf{x}) + \mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} - \tau\left[\delta^2 - \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{p}\right], \tag{1.92}$$

where τ is the Lagrange multiplier associated with the trust-region inequality constraint. Now the corresponding necessary condition is just

$$\nabla_p L_T(\mathbf{p},\tau) = \mathbf{g} + \mathbf{H}\mathbf{p} + \tau\mathbf{p} = \mathbf{g} + [\mathbf{H} + \tau\mathbf{I}]\,\mathbf{p} = \mathbf{0}. \tag{1.93}$$

There are two possible solutions to this problem. If the trust-radius constraint is inactive, then $\tau = 0$ and the search direction $\mathbf{p}$ is just the unmodified Newton direction. On the other hand, if the trust-radius constraint is active, $\|\mathbf{p}\| = \delta$ and the multiplier $\tau > 0$. In fact, the trust radius δ and the multiplier τ are implicitly (and inversely) related to each other—when δ is large, τ is small, and vice versa. Traditional trust-region methods

maintain a current estimate of the trust radius δ and adjust the parameter τ such that the constraint $\|\mathbf{p}\| = \delta$ is approximately satisfied. An alternative is to view the parameter τ as a way to construct a modified Hessian matrix $\mathbf{H} + \tau\mathbf{I}$. This interpretation was suggested by Levenberg [83] for nonlinear least squares (NLS) problems, and we shall often refer to τ as the *Levenberg parameter*. Notice that as τ becomes large, the search direction approaches the gradient direction. Table 1.5 summarizes the limiting behavior of these quantities.

$\tau \to \infty$	$\tau \to 0$
$\delta \to 0$	$\delta \to \infty$
$\mathbf{p} \propto -\mathbf{g}$	$\mathbf{p} \to -\mathbf{H}^{-1}\mathbf{g}$
$\|\mathbf{p}\| \to 0$	$\|\mathbf{p}\| \to \|\mathbf{H}^{-1}\mathbf{g}\|$

Table 1.5: Trust-region behavior.

Although the trust-region approach has been introduced as a globalization technique for an unconstrained optimization problem, the basic idea is far more general. The trust-region concepts can be extended to constrained optimization problems as well as, obviously, to least squares and root-solving applications. Other definitions of the trust region itself using alternative norms have been suggested. Furthermore, constrained applications have been formulated with more than one trust region to reflect the conflicting aims of constraint satisfaction and objective function reduction. Finally, it should be emphasized that trust-region concepts are often used in conjunction with other globalization ideas. For example, a common way to modify the trust radius from one iteration to the next is to compare the values of the predicted and actual reduction in a suitable merit function.

1.11.4 Filters

When solving a constrained optimization problem, it is necessary to introduce some quantitative method for deciding that a Newton iterate is working. One approach for achieving this is to introduce a merit function that combines the conflicting goals of constraint satisfaction and objective function reduction. However, merit functions must explicitly assign some relative weight to each conflicting goal, and choosing the correct penalty weight is often difficult. This dilemma was illustrated by the results summarized in Table 1.3 and Table 1.4. Recently, Fletcher and Leyffer [54] have introduced an approach that will accept a Newton step if *either* the objective function *or* the constraint violation is decreased. The filter approach recognizes that there are two conflicting aims in nonlinear programming. The first is to minimize the objective function, that is, choose $\mathbf{x}$ to minimize

$$F(\mathbf{x}). \tag{1.94}$$

The second is to choose $\mathbf{x}$ to minimize the constraint violation

$$v[\mathbf{c}(\mathbf{x})], \tag{1.95}$$

where the *violation* can be measured using any suitable norm. Fletcher and Leyffer define $v[\mathbf{c}(\mathbf{x})] \equiv \|\tilde{\mathbf{c}}\|_1$, where $\tilde{c}_k = \min(0, c_k)$, for inequalities of the form $c_k \geq 0$. The basic idea is to compare information from the current iteration to information from previous iterates and then "filter" out the bad iterates.

To formalize this, denote the values of the objective and constraint violation at the point $\mathbf{x}^{(k)}$ by

$$\{F^{(k)}, v^{(k)}\} \equiv \{F(\mathbf{x}^{(k)}), v[\mathbf{c}(\mathbf{x}^{(k)})]\}. \tag{1.96}$$

When comparing the information at two different points $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(j)}$, a pair $\{F^{(k)}, v^{(k)}\}$ is said to *dominate* another pair $\{F^{(j)}, v^{(j)}\}$ if and only if both $F^{(k)} \leq F^{(j)}$ and $v^{(k)} \leq v^{(j)}$. Using this definition, we can then define a *filter* as a list of pairs

$$\mathcal{F} = \begin{bmatrix} F^{(1)}, v^{(1)} \\ F^{(2)}, v^{(2)} \\ \vdots \\ F^{(K)}, v^{(K)} \end{bmatrix} \tag{1.97}$$

such that no pair dominates any other. A new point $\{F^{(\ell)}, v^{(\ell)}\}$ is said to be acceptable for inclusion in the filter if it is not dominated by any point in the filter.

Example 1.7. To illustrate how this procedure works, let us revisit Example 1.6, posed as the following optimization problem: Minimize

$$F(\mathbf{x}) = (1 - x_1)^2$$

subject to

$$\mathbf{c}(\mathbf{x}) = 10(x_2 - x_1^2) = 0.$$

Table 1.6 summarizes the behavior of a filter algorithm using the same sequence of iterates as in Table 1.4. The problem is illustrated in Figure 1.8. The iteration begins at $\mathbf{x}^{\mathsf{T}} = (-1.2, 1)$, and the first filter entry is $(F^{(1)}, v^{(1)}) = (4.84, 4.4)$. The dark shaded region in Figure 1.8 defines points that would be rejected by the filter because both the objective function and constraint violation would be worse than the first filter entry values. The second iteration at the point $\mathbf{x}^{\mathsf{T}} = (1, -3.84)$ is *accepted* by the filter because the objective function is better even though the constraint violation is worse. This point is added to the filter, thereby defining a new "unacceptable" region, which is the union of the regions defined by each point in the filter (i.e., the dark and light shaded regions). The final iterate does not violate the regions defined by either of the filter entries and is accepted as the solution.

Iter.	x_1	x_2	$F(\mathbf{x})$	$v[\mathbf{c}(\mathbf{x})]$
0	−1.2000	1.0000	4.84	4.4
1	1.0000	−3.8400	0	48.4
2	1.0000	1.0000	0	0

Table 1.6: Filter iteration summary.

Observe that the filter only provides a mechanism for accepting or rejecting an iterate. It does not suggest how to correct the step if a point is rejected by the filter. Fletcher and Leyffer use the filter in conjunction with a trust-region approach. On the other hand, a line-search technique that simply reduces the steplength is also an acceptable method for correcting the iterate. Practical implementation of the filter mechanism also must preclude a sequence of points that becomes unbounded in either $F(\mathbf{x})$ or $v[\mathbf{c}(\mathbf{x})]$.

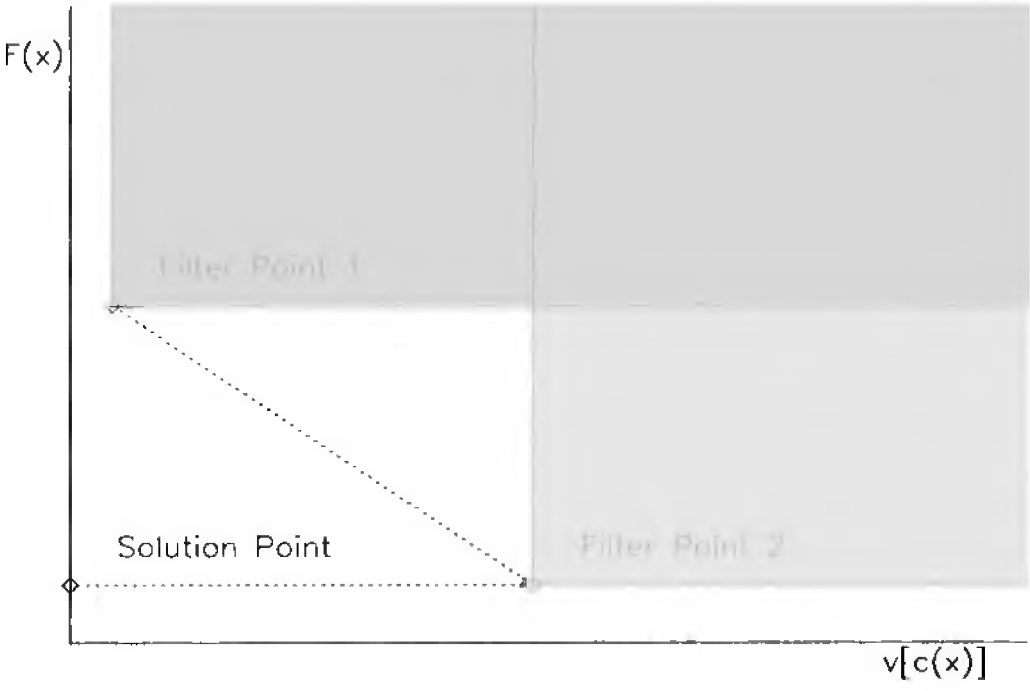


Figure 1.8: NLP filter.

A generous overestimate of the upper bound on $F(\mathbf{x})$ can be included as an additional "northwest corner" entry in the filter. Similarly, an absolute upper limit on the constraint violation can be included as a "southeast corner" entry in the filter. Computation of these extreme values is readily accommodated by including the value of the largest Lagrange multiplier for each iterate in the list of saved information. The amount of information saved in the filter list is usually rather small because, when a new entry $\{F^{(\ell)}, v^{(\ell)}\}$ is added, all points dominated by the new entry are deleted from the filter.

1.12 Nonlinear Programming

The general nonlinear programming (NLP) problem can be stated as follows: Find the n-vector $\mathbf{x}^{\mathsf{T}} = (x_1, \ldots, x_n)$ to *minimize* the scalar objective function

$$F(\mathbf{x}) \tag{1.98}$$

subject to the m constraints

$$\mathbf{c}_L \leq \mathbf{c}(\mathbf{x}) \leq \mathbf{c}_U \tag{1.99}$$

and the simple bounds

$$\mathbf{x}_L \leq \mathbf{x} \leq \mathbf{x}_U. \tag{1.100}$$

Equality constraints can be imposed by setting $\mathbf{c}_L = \mathbf{c}_U$.

The KKT necessary conditions for $\mathbf{x}^*$ to be an optimal point require that

- $\mathbf{x}^*$ be feasible, i.e., (1.99) and (1.100) are satisfied;
- the Lagrange multipliers $\boldsymbol{\lambda}$ corresponding to (1.99) and $\boldsymbol{\nu}$ corresponding to (1.100) satisfy

$$\mathbf{g} = \mathbf{G}^T\boldsymbol{\lambda} + \boldsymbol{\nu}; \tag{1.101}$$

- the Lagrange multipliers for the inequalities be

$$\begin{cases} \text{nonpositive} & \text{for active upper bounds,} \\ \text{zero} & \text{for strictly satisfied constraints,} \\ \text{nonnegative} & \text{for active lower bounds;} \end{cases}$$

- the Jacobian $\widetilde{\mathbf{G}}$ corresponding to the active constraints have full row rank (the *constraint qualification test*).

1.13 An SQP Algorithm

Let us now outline the basic steps of a sequential programming or SQP method for solving the general NLP problem. SQP methods are among the most widely used algorithms for solving general nonlinear programs, and there are many variations of the basic approach. The method described is implemented in the SOCS [25] software and is similar to the algorithm employed by the NPSOL [60] software. For additional information on SQP methods, see, for example, Fletcher [53] and Gill, Murray, and Wright [63].

The basic approach described in Section 1.8 was to introduce a quadratic approximation to the Lagrangian and a linear approximation to the constraints. This development lead to (1.67) and (1.68). An identical approach is followed here. Assume that the iteration begins at the point $(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$. The fundamental idea is to solve the following QP subproblem:

Compute $\mathbf{p}$ to minimize a quadratic approximation to the Lagrangian

$$\mathbf{g}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p} \tag{1.102}$$

subject to the linear approximation to the constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U \tag{1.103}$$

with bound vectors defined by

$$\mathbf{b}_L = \begin{bmatrix} \mathbf{c}_L - \mathbf{c} \\ \mathbf{x}_L - \mathbf{x} \end{bmatrix}, \qquad \mathbf{b}_U = \begin{bmatrix} \mathbf{c}_U - \mathbf{c} \\ \mathbf{x}_U - \mathbf{x} \end{bmatrix}. \tag{1.104}$$

The solution of this QP subproblem defines more than just the search direction $\mathbf{p}$. In particular, the quadratic program determines an estimate of the active set of constraints and an estimate of the corresponding Lagrange multipliers $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$. Thus, it is possible to define search directions for the multipliers

$$\Delta\boldsymbol{\lambda} \equiv \widehat{\boldsymbol{\lambda}} - \boldsymbol{\lambda}, \tag{1.105}$$

$$\Delta\boldsymbol{\nu} \equiv \widehat{\boldsymbol{\nu}} - \boldsymbol{\nu}. \tag{1.106}$$

Finally, the QP solution can be used to construct information needed to compute a merit function. A linear prediction for the value of the constraints is given by

$$\bar{\mathbf{s}} \equiv \mathbf{G}\mathbf{p} + \mathbf{c}. \tag{1.107}$$

Assuming we begin the iteration with an estimate for the *slack variables* $\mathbf{s}$, then it follows that

$$\Delta\mathbf{s} \equiv \bar{\mathbf{s}} - \mathbf{s} = \mathbf{G}\mathbf{p} + (\mathbf{c} - \mathbf{s}). \tag{1.108}$$

The term $(\mathbf{c} - \mathbf{s})$ has intentionally been isolated in this expression because it measures the "deviation from linearity" in the constraints $\mathbf{c}$. Notice that if the constraints $\mathbf{c}(\mathbf{x})$ are linear functions of the variables $\mathbf{x}$, then the term $(\mathbf{c} - \mathbf{s}) = \mathbf{0}$. Now the augmented search direction formed by collecting these expressions is given by

$$\begin{bmatrix} \bar{\mathbf{x}} \\ \bar{\boldsymbol{\lambda}} \\ \bar{\boldsymbol{\nu}} \\ \bar{\mathbf{s}} \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \\ \mathbf{s} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{p} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{\nu} \\ \Delta\mathbf{s} \end{bmatrix}. \tag{1.109}$$

As before, the scalar α defines the steplength. Observe that when a full Newton step is taken ($\alpha = 1$), the new NLP Lagrange multipliers are just the QP multipliers, i.e., $\bar{\boldsymbol{\lambda}} = \hat{\boldsymbol{\lambda}}$ and $\bar{\boldsymbol{\nu}} = \hat{\boldsymbol{\nu}}$.

Like all Newton methods, it is necessary to incorporate some type of globalization strategy. To this end, Gill, Murray, Saunders, and Wright [59] suggest a modified form of the augmented Lagrangian merit function (1.85) given by

$$M(\mathbf{x}, \boldsymbol{\lambda}, \mathbf{s}) = F - \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{c} - \mathbf{s}) + \frac{1}{2}(\mathbf{c} - \mathbf{s})^{\mathsf{T}}\boldsymbol{\Theta}(\mathbf{c} - \mathbf{s}), \tag{1.110}$$

where $\boldsymbol{\Theta}$ is a diagonal matrix of penalty weights. They also prove convergence for an SQP algorithm using this merit function, provided the SQP subproblem has a solution. This requires bounds on the derivatives and condition number of the Hessian matrix. Thus, the steplength α must be chosen to satisfy a sufficient decrease condition of the form (1.89). Most robust implementations incorporate a safeguarded line-search algorithm using quadratic and/or cubic polynomial interpolation. In addition, it is necessary that the penalty weights $\boldsymbol{\Theta}$ be chosen such that

$$M'(0) \leq -\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}. \tag{1.111}$$

As a practical matter, this single condition does not uniquely define all of the penalty weights in the matrix $\boldsymbol{\Theta}$, and so a minimum norm estimate is used. Estimates for the slack variables $\mathbf{s}$ are also needed to construct the slack direction (1.108) and these quantities can be computed by minimizing M as a function of $\mathbf{s}$ before taking the step. We postpone details of these calculations to the next chapter.

Example 1.8. To illustrate the behavior of an SQP method, let us minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2 + \log(x_1 x_2) \tag{1.112}$$

subject to the constraint

$$c_1(\mathbf{x}) = x_1 x_2 \geq 1 \tag{1.113}$$

with bounds

$$0 \leq x_1 \leq 10, \tag{1.114}$$
$$0 \leq x_2 \leq 10. \tag{1.115}$$

Assume that $\mathbf{x}^{(0)} = (0.5, 2)^\mathsf{T}$ is an initial guess for the solution.

Figure 1.9 illustrates the iteration history when the SQP algorithm implemented in SOCS is applied to this problem. Notice that it is not possible to evaluate the objective function at the first predicted point $(1, 0)$. This is considered a "function error" by the SOCS software, and the response in the line search is to reduce the steplength. In this case, $\alpha = 9.04543273 \times 10^{-2}$ produces the point $(0.545227, 1.81909)$. Subsequent steps proceed to the solution at $(1, 1)$.

1.14 What Can Go Wrong

The material in this chapter has been presented to give the reader an understanding of how a method *should* work in practice. The discussion is designed to help users efficiently use high-quality optimization software to solve practical problems. However, in practice, the user can expect things to go wrong! In this section, we describe a number of common difficulties that can be encountered and suggest remedial action to correct the deficiencies. For ease of presentation, it is convenient to discuss the difficulties individually. Of course, real applications may involve more than a single difficulty and the user must be prepared to correct all problems before obtaining satisfactory performance from optimization software.

1.14.1 Infeasible Constraints

One of the most common difficulties encountered occurs when the NLP problem has infeasible constraints. To be more precise, infeasible constraints have *no* solution.

Example 1.9. For example, the following constraints have no feasible region.

$$\begin{aligned} c_1(\mathbf{x}) &= x_1^2 x_2 - 1 \geq 0, \\ c_2(\mathbf{x}) &= x_2 \leq -\tfrac{1}{10}. \end{aligned} \tag{1.116}$$

When general-purpose NLP software is applied to such a problem, it is likely that one or more of the following symptoms will be observed:

- the QP subproblem has no solution, which can occur if the linearized constraints have no solution;
- many NLP iterations produce very little progress;

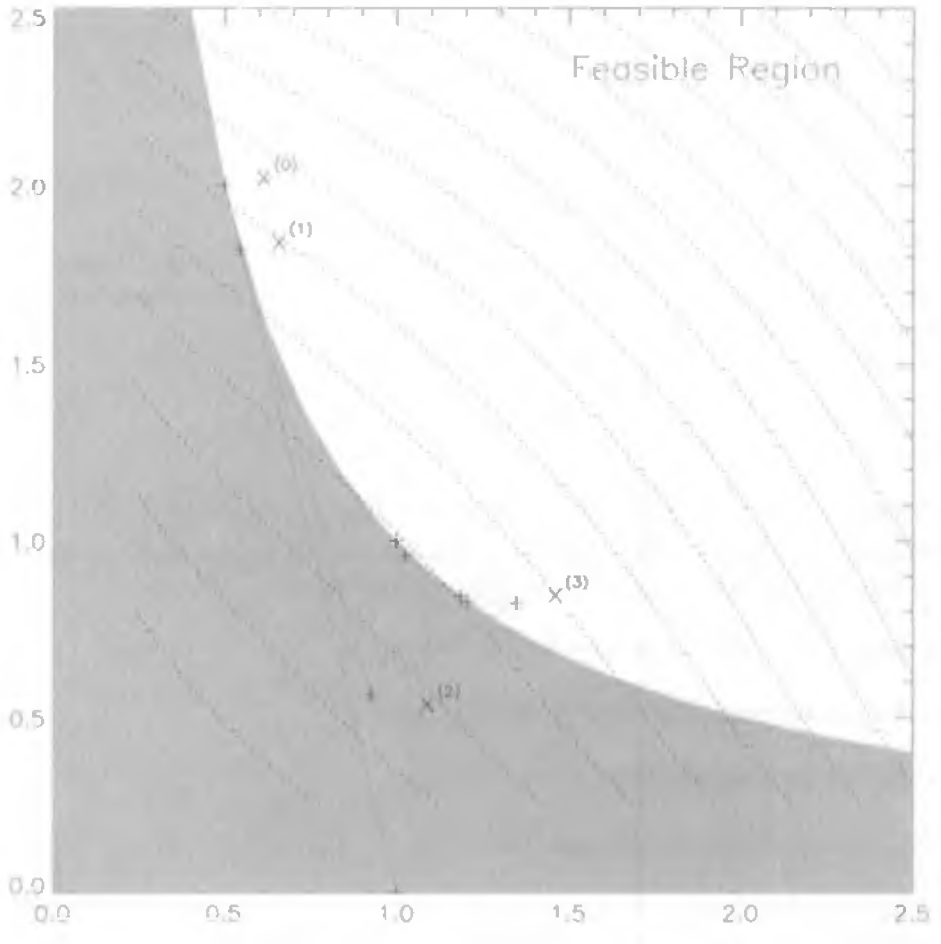


Figure 1.9: NLP example.

- the penalty parameters Θ become very large;
- the condition number of the projected Hessian matrix becomes large: or
- the Lagrange multipliers become large.

Although most robust software will attempt to "gracefully" detect this situation, ultimately the only remedy is to reformulate the problem!

1.14.2 Rank-Deficient Constraints

In contrast to the previous situation, it is possible that the constraints are consistent, that is, they do have a solution. However, at the solution, the Jacobian matrix may be either ill-conditioned or rank deficient.

Example 1.10. For example, consider minimizing

$$F(\mathbf{x}) = (x_1 - 2)^2 + x_2^2 \tag{1.117}$$

subject to the constraints

$$\begin{aligned} c_1(\mathbf{x}) &= x_1 \geq 0, \\ c_2(\mathbf{x}) &= x_2 \geq 0, \\ c_3(\mathbf{x}) &= (1-x_1)^3 - x_2 \geq 0. \end{aligned} \tag{1.118}$$

The solution to this problem is $\mathbf{x}^* = (1,0)^\mathsf{T}$ and, clearly, $\mathbf{c}(\mathbf{x}^*) = \mathbf{0}$. However, at the solution, the active set is $\mathcal{A}^* = \{c_2, c_3\}$ and the Jacobian matrix corresponding to these constraints is rank deficient, i.e.,

$$\widetilde{\mathbf{G}} = \begin{bmatrix} 0 & 1 \\ -3(1-x_1)^2 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -1 \end{bmatrix}. \tag{1.119}$$

This example violates the constraint-qualification test.

When general-purpose NLP software is applied to such a problem, it is likely that one or more of the following symptoms will be observed:

- many NLP iterations produce very little progress;
- the penalty parameters $\boldsymbol{\Theta}$ become very large;
- the Lagrange multipliers become large; or
- the rank deficiency in $\widetilde{\mathbf{G}}$ is detected.

Unfortunately, detecting rank deficiency in the Jacobian is not a straightforward numerical task! Consequently, it is quite common to confuse this difficulty with inconsistent constraints. Again, the remedy is to reformulate the problem!

1.14.3 Constraint Redundancy

A third type of difficulty can occur when the problem formulation contains constraints that are redundant. Thus, although the constraints have a solution, they may be unnecessary to the problem formulation. The following two examples illustrate the situation.

Example 1.11. Minimize

$$F(\mathbf{x}) = x_1^2 + x_2^2 \tag{1.120}$$

subject to the constraint

$$c_1(\mathbf{x}) = x_1 - x_2 = 0. \tag{1.121}$$

In this case, the unconstrained solution is $\mathbf{x}^* = (0,0)^\mathsf{T}$, and the constraint is trivially satisfied. Constraint c_1 can be eliminated from the formulation without changing the answer.

Example 1.12. Minimize

$$F(\mathbf{x}) = (x_1 - 2)^2 + x_2^2 + x_3^2 \tag{1.122}$$

subject to the constraints

$$\begin{aligned} c_1(\mathbf{x}) &= x_1 + x_2 - 1 &&= 0, \\ c_2(\mathbf{x}) &= 2x_1 + 2x_2 - 2 &&= 0. \end{aligned} \tag{1.123}$$

In this case, constraint $c_2 = 2c_1$ and, clearly, one of the constraints is redundant. Note also that the Jacobian matrix $\widetilde{\mathbf{G}}$ is rank deficient.

Symptoms indicating redundancy such as in Example 1.11 include

- Lagrange multipliers near zero and
- difficulty detecting the active set.

On the other hand, constraint redundancy such as in Example 1.12 is likely to exhibit symptoms similar to the rank-deficient cases discussed previously. Obviously, the remedy is to reformulate the problem and eliminate the redundant constraints!

1.14.4 Discontinuities

Perhaps the single biggest obstacle encountered in the practical application of NLP methods is the presence of discontinuous behavior. All of the numerical methods described assume that the objective and constraint functions are continuous and differentiable. When discontinuities are present in the problem functions, the standard quadratic/linear models that form the basis for most NLP methods are no longer appropriate. In fact, the KKT necessary conditions do not apply!

Unfortunately, in practice, there are many common examples of discontinuous behavior. Typical sources include

- branching caused by IF tests in code;
- absolute value, max, and min functions;
- linear interpolation of tabular data; and
- "internal" iteration loops, e.g., adaptive quadrature, root finding.

The most common symptoms of discontinuous functions are

- slow convergence or divergence,
- small steps ($\alpha \approx 0$) in the line search, and
- possible ill-conditioning of the Hessian matrix.

Example 1.13. Treating absolute values.

Bad Formulation

Find (x, y) to minimize

$$|x|$$

subject to

$$y - x^2 = 1.$$

Good Formulation

Find (x_1, x_2, y) to minimize

$$x_1 + x_2$$

subject to

$$y - (x_1 - x_2)^2 = 1,$$
$$x_1 \geq 0,$$
$$x_2 \geq 0.$$

The "trick" used is to replace $x \rightarrow (x_1 - x_2)$ and then observe that $|x| \rightarrow (x_1 + x_2)$. The optimal solution for the preferred formulation is $\mathbf{x}^* = (0, 0, 1)$. As expected, the preferred formulation is solved by SOCS in 47 evaluations. In contrast, the original formulation requires 117 function evaluations and terminates with a small step warning.

Example 1.14. Minimizing maximum values (*minimax problems*).

Bad Formulation

Find (x_1, x_2) to minimize

$$\max_k |y_k - d_k|$$

for $k = 1, 2, 3$, where

$$y_k = x_1 + x_2 k,$$
$$d = (1, 1.5, 1.2).$$

Good Formulation

Find (x_1, x_2, s) to minimize

$$s$$

for $k = 1, 2, 3$, where

$$y_k - d_k - s \leq 0,$$
$$y_k - d_k + s \geq 0.$$

The trick here is to introduce the slack variable s as an absolute bound on the quantities being minimized, i.e., $|y_k - d_k| \leq s$. The absolute value function is then eliminated since $-s \leq y_k - d_k \leq s$ can be rewritten as two inequality constraints. The optimal solution for the preferred formulation is $\mathbf{x}^* = (1.1, 0.1, 0.2)$. As expected, the preferred formulation is solved by SOCS in 30 evaluations. In contrast, the original formulation terminates with a small step warning after 271 evaluations at the point $\mathbf{x}^* = (1.09963, 0.100148)$.

In subsequent chapters, a great deal of attention will be given to the correct formulation of optimal control applications such that discontinuous behavior can be avoided. Nevertheless, it is worth mentioning that specific remedial action for these problems includes

- recoding and/or reformulating to eliminate branching, absolute value, max, and min problems (cf. Examples 1.13 and 1.14);
- interpolation or approximation of tabular data using functions with continuity through second derivatives (cf. Example 5.3);
- avoiding the use of variable-step, variable-order integration when optimizing; and

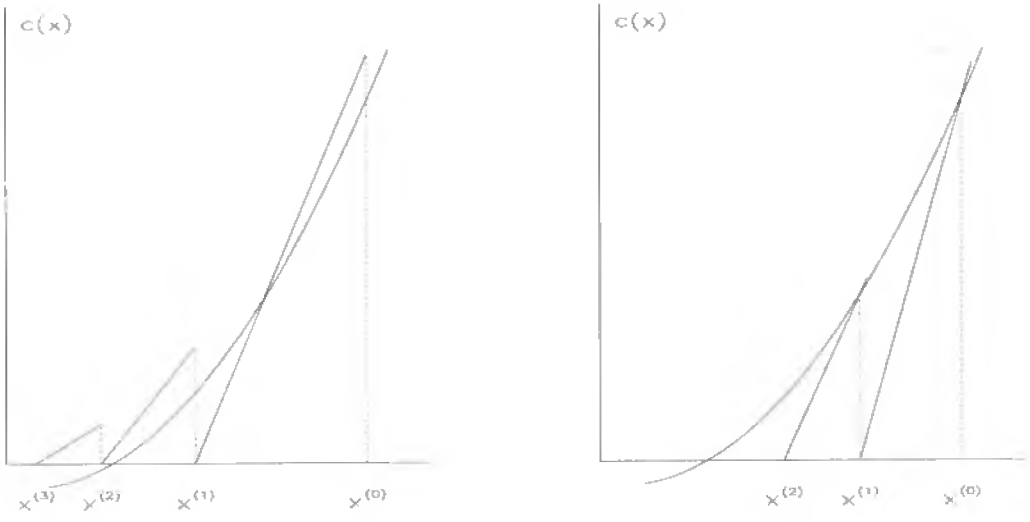


Figure 1.10: The effects of noise.

- reformulation of "internal" iterations (e.g., Kepler's equation) using additional "external" NLP constraints (cf. Example 5.5).

Because discontinuous behavior plays such a major role in the success or failure of an NLP application, it is worthwhile to understand the effects of "noise" on the Newton iteration. Figure 1.10 illustrates a typical Newton iteration in the presence of two types of noise. Let us assume that we are trying to solve a constraint $c(x) = 0$. However, because of errors in the function evaluation, what the Newton method "sees" is the contaminated result $c(x) + \epsilon$. Because the intent is to drive the constraint to zero, ultimately the value will be "swamped" by the noise ϵ and we expect that the iteration will *not* converge! This situation is shown in the left portion of Figure 1.10. On the other hand, suppose that the value of the constraint is correct but the slope is contaminated by noise as illustrated in the right portion of Figure 1.10. Thus, instead of computing the true slope $c'(x)$, the Newton iterate actually uses the value $c'(x) + \epsilon$. In this case, we can expect the iteration to locate a solution, but at a degraded *rate* of convergence. We can carry this analysis further by viewing optimization as equivalent to solving $\nabla_x L = 0$. Then, by analogy, if the gradient is contaminated by noise, $\nabla_x L + \epsilon$, the iteration will not converge, because ultimately the true gradient will be dominated by the noise. On the other hand, if the Hessian information is noisy, $\nabla^2_{xx} L + \epsilon$, we expect the *rate* of convergence to be degraded. It is worth noting that an approximation to the slope (e.g., secant or quasi-Newton) in the absence of noise changes the Newton iterates in a similar fashion.

1.14.5 Scaling

Scaling affects everything! Poor scaling can make a good algorithm bad. Scaling changes the convergence rate, termination tests, and numerical conditioning. The most common way to scale a problem is to introduce variable scaling of the form

$$\tilde{x}_k = u_k x_k + r_k \tag{1.124}$$

for $k = 1, \ldots, n$. In this expression, the scaled variables $\tilde{x}_k$ are related to the unscaled variables by the variable scale weights u_k and shifts r_k. In like fashion, the objective and constraints are commonly scaled using

$$\tilde{c}_k = w_k c_k, \tag{1.125}$$

$$\tilde{F} = w_0 F \tag{1.126}$$

for $k = 1, \ldots, m$. The intent is to let the optimization algorithm work with the "well-scaled" quantities $\tilde{x}$, $\tilde{c}$, and $\tilde{F}$ in order to improve the performance of the algorithm. Unfortunately, it is not at all clear what it means to be well scaled!

Nevertheless, conventional wisdom suggests that some attempt should be made to construct a well-scaled problem and, consequently, we give the following *hints* (*not rules*) for good scaling:

- normalize the independent variables to have the same "range," e.g.,

$$0 \leq \tilde{x}_k \leq 1;$$

- normalize the dependent functions to have the same magnitude, i.e.,

$$F \approx c_1 \approx c_2 \approx \cdots \approx c_m \approx 1;$$

- normalize the rows and columns of the Jacobian to be of the same magnitude;
- scale the dependent functions so that the Lagrange multipliers are one:

$$|\lambda_1| \approx |\lambda_2| \approx \cdots \approx |\lambda_m| \approx 1;$$

- scale the problem so that the condition number of the projected Hessian matrix is close to one;
- scale the problem so that the condition number of the KKT matrix (1.66) is close to one.

Constructing an automatic computational procedure that will meet all of these goals is probably not possible. Furthermore, even if a strategy could be devised that produces "good scaling" at one point, say $\mathbf{x}$ for nonlinear functions, this may be "bad scaling" at another point $\overline{\mathbf{x}}$. As a practical matter in the SOCS software, we attempt to produce a well-scaled Jacobian at the user-supplied initial point, but also allow the user to override this scaling by input. For more helpful discussion on this subject, the reader is referred to [63, Chapter 8].

Chapter 2

Large, Sparse Nonlinear Programming

2.1 Overview: Large, Sparse NLP Issues

Chapter 1 presents background on nonlinear programming (NLP) methods that are applicable to most problems. In this chapter, we will focus on NLP methods that are appropriate for problems that are both *large* and *sparse*. To set the stage for what follows, it is useful to define what we mean by large and sparse. The definition of "large" is closely tied with the tools available for solving linear systems of equations. For our purposes, we will consider a problem "small" if the underlying linear systems can be solved using a dense, direct matrix factorization. Thus, for today's computers, this suggests that the problem size is probably limited to matrices of order $n \approx 1000$. If the linear equations are solved using methods that exploit sparsity in the matrices but still use direct matrix factorizations, then with current computer hardware, the upper limit on the size of the problem is $n \approx 10^6$. This problem is considered "large" and is the focus of methods in this book. Although some of the NLP methods can be extended, as a rule we will not address "huge" problems for which $n > 10^6$. Typically, linear systems of this size are solved by iterative (as opposed to direct) methods. The second discriminator of interest concerns matrix sparsity. A matrix is said to be "sparse" when many of the elements are zero. For most applications of interest, the number of nonzero elements in the Hessian matrix $\mathbf{H}$ and Jacobian matrix $\mathbf{G}$ is less than 1%.

Many of the techniques described in Chapter 1 extend, without change, to large, sparse problems. On the other hand, there are some techniques that cannot be used for large, sparse applications. In this chapter, we will focus on those issues that are unique to the large, sparse NLP problem.

The first item concerns how to calculate Jacobian and Hessian information for large, sparse problems. All of the Newton-based methods described in Chapter 1 rely on the availability of first and second derivative information. For most practical applications, first derivatives are computed using finite difference approximations. Unfortunately, a single finite difference gradient evaluation can require n additional function evaluations, and when n is large, this computational cost can be excessive. Furthermore, computing second derivatives by naive extensions of the methods described in Chapter 1 becomes unattractive for a number of reasons. First, maintaining a sparse quasi-Newton approx-

imation that is also positive definite appears unpromising. Second, even if we accept an approximation that is indefinite, one can expect to spend $\mathcal{O}(n)$ iterations before a "good" Hessian approximation is constructed—and if n is large, the number of iterations can be excessive! To overcome these drawbacks, current research has focused on *pointwise quasi-Newton updates* [81], [82] and *limited memory updates*. Another alternative is to abandon the quasi-Newton approach altogether and construct this information using sparse finite differences. This technique will be explained in the next section.

The second major item addressed in this chapter concerns how to efficiently construct a Newton step when the underlying matrices are large and sparse. We first concentrate on a sparse QP algorithm and present a detailed description of the method. We then address how the method can be extended to the important sparse nonlinear least squares problem.

2.2 Sparse Finite Differences

2.2.1 Background

In general, let us define the first derivatives of a set of v functions $q_i(\mathbf{x})$ with respect to n variables $\mathbf{x}$ by the $v \times n$ matrix

$$\mathbf{D} \equiv \begin{bmatrix} (\nabla q_1)^\top \\ (\nabla q_2)^\top \\ \vdots \\ (\nabla q_v)^\top \end{bmatrix} = \frac{\partial \mathbf{q}}{\partial \mathbf{x}}. \tag{2.1}$$

It will also be of interest to compute second derivatives of a linear combination of the functions. In particular, we define the second derivatives of the function

$$\Omega(\mathbf{x}) = \sum_{i=1}^{v} \omega_i q_i(\mathbf{x}) \tag{2.2}$$

with respect to n variables $\mathbf{x}$ by the $n \times n$ matrix

$$\mathbf{E} \equiv \nabla^2 \Omega(\mathbf{x}) = \sum_{i=1}^{v} \omega_i \nabla^2 q_i(\mathbf{x}). \tag{2.3}$$

The notion of sparse finite differencing was introduced by Curtis, Powell, and Reid [40]. They proposed that the columns of $\mathbf{D}$ be partitioned into subsets (*index sets*) Γ^k such that each subset has at most one nonzero element per *row*. Then define the perturbation direction vector by

$$\mathbf{\Delta}^k = \sum_{j \in \Gamma^k} \delta_j \mathbf{e}_j, \tag{2.4}$$

where δ_j is the perturbation size for variable j and $\mathbf{e}_j$ is a unit vector in direction j. Using this partitioning, it can be demonstrated that the central difference estimates of the first derivatives for $i = 1, \ldots, v$ and $j \in \Gamma^k$ are

$$\mathbf{D}_{ij} \approx \frac{1}{2\delta_j}[q_i(\mathbf{x} + \mathbf{\Delta}^k) - q_i(\mathbf{x} - \mathbf{\Delta}^k)]. \tag{2.5}$$

In a similar fashion, second derivative estimates for $i \in \Gamma^k$ and $j \in \Gamma^\ell$ are

$$\mathbf{E}_{ij} \approx \frac{1}{\delta_i \delta_j}[\Omega(\mathbf{x} + \boldsymbol{\Delta}^k + \boldsymbol{\Delta}^\ell) + \Omega(\mathbf{x}) - \Omega(\mathbf{x} + \boldsymbol{\Delta}^k) - \Omega(\mathbf{x} + \boldsymbol{\Delta}^\ell)] \tag{2.6}$$

and

$$\mathbf{E}_{ii} \approx \frac{1}{\delta_i^2}[\Omega(\mathbf{x} + \boldsymbol{\Delta}^k) + \Omega(\mathbf{x} - \boldsymbol{\Delta}^k) - 2\Omega(\mathbf{x})]. \tag{2.7}$$

Denote the total number of index sets Γ^k needed to span the columns of $\mathbf{D}$ by γ. Now observe that by using the same index sets for the first and second derivatives, it is possible to compute

1. central difference first derivatives using 2γ perturbations and
2. first and second derivatives using $\gamma(\gamma + 3)/2$ perturbations.

Since a function evaluation can be costly, it is clear that the number of index sets γ determines the cost of constructing this derivative information. It can be demonstrated that the maximum number of nonzeros in any row of $\mathbf{D}$ is a lower bound on the number γ. Coleman and Moré [39] have also shown that computing the smallest number of index sets is a graph-coloring problem. For most applications, acceptable approximations to the minimum number of index sets can be achieved using the "greedy algorithm" suggested by Curtis, Powell, and Reid [40]. Regardless of how the index sets are constructed, the important point is that for the optimal control problems of interest, $\gamma \ll n$. In simple terms, the number of perturbations is much smaller than the number of variables.

There are two computational issues that should be emphasized with regard to a sparse differencing implementation. First, from direct inspection of (2.3), it might appear that storage is needed for all of the individual Hessian matrices $\nabla^2 q_i(\mathbf{x})$. This is not true! In fact, the calculations can be organized such that the summation is done first, thereby making it necessary to only store the result. Second, this procedure uses a single perturbation size to construct difference approximations for many functions. As such, choosing the "best" perturbation to balance truncation and roundoff errors for all of the functions is a compromise, and some inaccuracy can be expected. Nevertheless, for well-scaled functions, this is not a significant issue, especially because the central difference gradient information given by (2.5) is $\mathcal{O}(\delta^2)$.

2.2.2 Sparse Differences in Nonlinear Programming

When a finite difference method is used to construct the Jacobian, it is natural to identify the constraint functions as the quantities being differentiated in (2.1). In other words,

$$\mathbf{q} = \begin{bmatrix} \mathbf{c} \\ F \end{bmatrix} \tag{2.8}$$

and

$$\mathbf{D} = \begin{bmatrix} \mathbf{G} \\ \mathbf{g}^\mathsf{T} \end{bmatrix}. \tag{2.9}$$

It is also natural to define

$$\boldsymbol{\omega}^\mathsf{T} = (-\lambda_1, \ldots, -\lambda_m, 1), \tag{2.10}$$

where λ_k are the Lagrange multipliers with $v = m + 1$, so that (2.2)

$$\Omega(\mathbf{x}) = \sum_{i=1}^{v} \omega_i q_i(\mathbf{x}) = F - \sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}) = L(\mathbf{x}, \boldsymbol{\lambda}) \tag{2.11}$$

is the *Lagrangian* for the NLP problem. Clearly, it follows that the Hessian of the Lagrangian $\mathbf{H} = \mathbf{E}$, where $\mathbf{E}$ is given by (2.3).

2.3 Sparse QP Subproblem

The efficient solution of the sparse QP subproblem can be achieved using a method proposed by Gill, Murray, Saunders, and Wright [61, 62]. In general, the calculation of a step in a QP subproblem requires the solution of the KKT system (1.66) as described in Sections 1.8 and 1.10. For convenience, recall that the *QP subproblem* (1.102)–(1.103) is as follows:

Compute $\mathbf{p}$ to minimize a quadratic approximation to the Lagrangian

$$\mathbf{g}^{\mathsf{T}}\mathbf{p} + \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} \tag{2.12}$$

subject to the linear approximation to the constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U \tag{2.13}$$

with bound vectors defined by

$$\mathbf{b}_L = \begin{bmatrix} \mathbf{c}_L - \mathbf{c} \\ \mathbf{x}_L - \mathbf{x} \end{bmatrix}, \qquad \mathbf{b}_U = \begin{bmatrix} \mathbf{c}_U - \mathbf{c} \\ \mathbf{x}_U - \mathbf{x} \end{bmatrix}. \tag{2.14}$$

First, the QP subproblem (2.12) (2.13) is restated in the following *standard form.*

Compute $\widetilde{\mathbf{p}}$ to minimize

$$\widetilde{\mathbf{g}}^{\mathsf{T}}\widetilde{\mathbf{p}} + \frac{1}{2}\widetilde{\mathbf{p}}^{\mathsf{T}}\widetilde{\mathbf{H}}\widetilde{\mathbf{p}} \tag{2.15}$$

subject to the linear equality constraints

$$\widetilde{\mathbf{G}}\widetilde{\mathbf{p}} = \widetilde{\mathbf{b}} \tag{2.16}$$

and simple bounds

$$\widetilde{\mathbf{p}}_L \leq \widetilde{\mathbf{p}} \leq \widetilde{\mathbf{p}}_U. \tag{2.17}$$

Notice that this formulation involves only simple bounds (2.17) and equalities (2.16). This transformation can be accomplished by introducing *slack variables*, s_k. For example, a general inequality constraint of the form $\mathbf{a}_k^{\mathsf{T}}\mathbf{x} \geq b_k$ is replaced by the equality constraint $\mathbf{a}_k^{\mathsf{T}}\mathbf{x} + s_k = b_k$ and the bound $s_k \leq 0$. Notice that we have introduced the tilde notation to indicate that the original variables have been augmented to include the slacks. Observe that in this formulation, when a slack variable is "fixed" on an upper or lower bound, it is equivalent to the original inequality being in the active set. Thus, for a particular

QP iteration, the search direction in the "free" variables can be computed by solving the KKT system (1.66), which in this case is

$$\begin{bmatrix} \tilde{\mathbf{H}}_f & \tilde{\mathbf{G}}_f^T \\ \tilde{\mathbf{G}}_f & \mathbf{0} \end{bmatrix} \begin{bmatrix} -\tilde{\mathbf{p}}_f \\ \tilde{\boldsymbol{\lambda}} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{g}}_f \\ \mathbf{0} \end{bmatrix}. \tag{2.18}$$

We have used the f subscript to denote quantities corresponding to the "free" variables, and assume the iteration begins at a feasible point so that $\tilde{\mathbf{b}} = \mathbf{0}$. Let us define the large, sparse symmetric indefinite KKT matrix by

$$\mathbf{K}_0 = \begin{bmatrix} \tilde{\mathbf{H}}_f & \tilde{\mathbf{G}}_f^T \\ \tilde{\mathbf{G}}_f & \mathbf{0} \end{bmatrix}. \tag{2.19}$$

If the initial estimate of the active set is correct, the solution of the KKT system defines the solution of the QP problem. However, in general, it will be necessary to change the active set and solve a series of equality-constrained problems. In [61, 62], it is demonstrated that the solution to a problem with a new active set can be obtained by the symmetric addition of a row and column to the original $\mathbf{K}_0$, with a corresponding augmentation of the right-hand side. In fact, after k iterations, the KKT system is of dimension $n_0 + k$ and has the form

$$\begin{bmatrix} \mathbf{K}_0 & \mathbf{U} \\ \mathbf{U}^T & \mathbf{V} \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{z} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{w} \end{bmatrix}, \tag{2.20}$$

where $\mathbf{U}$ is $n_0 \times k$ and $\mathbf{V}$ is $k \times k$. The initial right-hand side of (2.18) is denoted by the n_0-vector $\mathbf{f}_0$. The k-vector $\mathbf{w}$ defines the additions to the right-hand side to reflect changes in the active set.

The fundamental feature of the method is that the system (2.20) can be solved using factorizations of $\mathbf{K}_0$ and $\mathbf{C}$, the $k \times k$ *Schur-complement* of $\mathbf{K}_0$:

$$\mathbf{C} \equiv \mathbf{V} - \mathbf{U}^T \mathbf{K}_0^{-1} \mathbf{U}. \tag{2.21}$$

Using the Schur-complement, the values for $\mathbf{y}$ and $\mathbf{z}$ are computed by solving in turn

$$\mathbf{K}_0 \mathbf{v}_0 = \mathbf{f}_0, \tag{2.22}$$

$$\mathbf{C}\mathbf{z} = \mathbf{w} - \mathbf{U}^T \mathbf{v}_0, \tag{2.23}$$

$$\mathbf{K}_0 \mathbf{y} = \mathbf{f} - \mathbf{U}\mathbf{z}. \tag{2.24}$$

Thus, each QP iteration requires one solve with the factorization of $\mathbf{K}_0$ and one solve with the factorization of $\mathbf{C}$. The solve for $\mathbf{v}_0$ only needs to be done once at the first iteration. Each change in the active set adds a new row and column to $\mathbf{C}$. It is relatively straightforward to update both $\mathbf{C}$ and its factorization to accommodate the change. It is important to keep $\mathbf{C}$ small enough to maintain a stable, dense factorization. This is achieved by refactoring the entire KKT matrix whenever $k > 100$. In general, the penalty for refactoring may be substantial. However, when the QP algorithm is used within the general NLP algorithm, it is possible to exploit previous estimates of the active set to give the QP algorithm a "warm start." In fact, as the NLP algorithm approaches a

solution, it is expected that the active set will be correctly identified and the resulting number of iterations k for the QP subproblem will become small.

The Schur-complement method derives its efficiency from two facts. First, the KKT matrix is factored only *once* using a very efficient *multifrontal algorithm* [3]. This software solves $\mathbf{Ax} = \mathbf{b}$ for $\mathbf{x}$, where $\mathbf{A}$ is an $n \times n$ real *symmetric indefinite* sparse matrix. Since $\mathbf{A}$ is symmetric, it can be factored as $\mathbf{A} = \mathbf{LDL}^{\mathsf{T}}$, where $\mathbf{L}$ is a unit lower-triangular matrix and $\mathbf{D}$ is a block-diagonal matrix composed solely of 1×1 and 2×2 blocks. Since $\mathbf{A}$ is not necessarily positive definite, pivoting to preserve stability is required. The package uses the threshold-pivoting generalization of Bunch and Kaufman with 2×2 block pivoting for sparse symmetric indefinite matrices. The software requires storage for the nonzero elements in the lower-triangular portion of the matrix and a work array. Second, subsequent changes to the QP active set can be computed using a *solve* with the previously factored KKT matrix and a *solve* with the small, dense Schur-complement matrix. Since the factorization of the KKT matrix is significantly more expensive than the solve operation, the overall method is quite effective. The algorithm is described in [20].

2.4 Merit Function

When a QP algorithm is used to approximate a general nonlinearly constrained problem, it may be necessary to adjust the steplength α in order to achieve "sufficient reduction" in a merit function as discussed in Section 1.11. The merit function we use is a modified version of (1.110), which was proposed by Gill, Murray, Saunders, and Wright in [59]. It is related to the function given by Rockafellar in [94]:

$$M(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}, \mathbf{s}, \mathbf{t}) = F - \boldsymbol{\lambda}^{\mathsf{T}}(\mathbf{c} - \mathbf{s}) - \boldsymbol{\nu}^{\mathsf{T}}(\mathbf{x} - \mathbf{t}) + \frac{1}{2}(\mathbf{c} - \mathbf{s})^{\mathsf{T}}\boldsymbol{\Theta}(\mathbf{c} - \mathbf{s}) + \frac{1}{2}(\mathbf{x} - \mathbf{t})^{\mathsf{T}}\boldsymbol{\Xi}(\mathbf{x} - \mathbf{t}). \tag{2.25}$$

The diagonal penalty matrices are defined by $\boldsymbol{\Theta}_{ii} = \theta_i$ and $\boldsymbol{\Xi}_{ii} = \xi_i$ for $\theta_i > 0$ and $\xi_i > 0$. The merit function is written to explicitly include terms for the bounds that were not present in the original formulation (1.110) [59]. For this merit function, the slack variables $\mathbf{s}$ and $\mathbf{t}$ at the beginning of a step are defined by

$$s_i = \begin{cases} c_{Li} & \text{if } c_{Li} > c_i - \lambda_i/\theta_i, \\ c_i - \lambda_i/\theta_i & \text{if } c_{Li} \le c_i - \lambda_i/\theta_i \le c_{Ui}, \\ c_{Ui} & \text{if } c_i - \lambda_i/\theta_i > c_{Ui}; \end{cases} \tag{2.26}$$

$$t_i = \begin{cases} x_{Li} & \text{if } x_{Li} > x_i - \nu_i/\xi_i, \\ x_i - \nu_i/\xi_i & \text{if } x_{Li} \le x_i - \nu_i/\xi_i \le x_{Ui}, \\ x_{Ui} & \text{if } x_i - \nu_i/\xi_i > x_{Ui}. \end{cases} \tag{2.27}$$

These expressions for the slack variables yield a minimum value for the merit function M for given values of the variables $\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}$ and penalty weights subject to the bounds on the slacks. The search direction in the real variables $\mathbf{x}$ is augmented to permit the

multipliers and the slack variables to vary according to

$$\begin{bmatrix} \overline{\mathbf{x}} \\ \overline{\boldsymbol{\lambda}} \\ \overline{\boldsymbol{\nu}} \\ \overline{\mathbf{s}} \\ \overline{\mathbf{t}} \end{bmatrix} = \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \\ \boldsymbol{\nu} \\ \mathbf{s} \\ \mathbf{t} \end{bmatrix} + \alpha \begin{bmatrix} \mathbf{p} \\ \Delta\boldsymbol{\lambda} \\ \Delta\boldsymbol{\nu} \\ \Delta\mathbf{s} \\ \Delta\mathbf{t} \end{bmatrix}. \tag{2.28}$$

The multiplier search directions, $\Delta\boldsymbol{\lambda}$ and $\Delta\boldsymbol{\nu}$, are defined using the QP multipliers $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$ according to

$$\Delta\boldsymbol{\lambda} \equiv \widehat{\boldsymbol{\lambda}} - \boldsymbol{\lambda} \tag{2.29}$$

and

$$\Delta\boldsymbol{\nu} \equiv \widehat{\boldsymbol{\nu}} - \boldsymbol{\nu}. \tag{2.30}$$

As in (1.107), the predicted slack variables are just

$$\overline{\mathbf{s}} = \mathbf{G}\mathbf{p} + \mathbf{c} = \mathbf{s} + \Delta\mathbf{s}. \tag{2.31}$$

Using this expression, the slack-vector step analogous to (1.108) is just

$$\Delta\mathbf{s} = \mathbf{G}\mathbf{p} + (\mathbf{c} - \mathbf{s}). \tag{2.32}$$

A similar technique defines the bound slack-vector search direction

$$\Delta\mathbf{t} = \mathbf{p} + (\mathbf{x} - \mathbf{t}). \tag{2.33}$$

Note that when a full step is taken ($\alpha = 1$), the updated estimates for the Lagrange multipliers $\overline{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{\nu}}$ are just the QP estimates $\widehat{\boldsymbol{\lambda}}$ and $\widehat{\boldsymbol{\nu}}$. The slack variables $\mathbf{s}$ and $\mathbf{t}$ are just the linear estimates of the constraints. The terms $(\mathbf{c} - \mathbf{s})$ and $(\mathbf{x} - \mathbf{t})$ in the merit function are measures of the *deviation* from linearity. In [59], Gill et al. prove global convergence for an SQP algorithm that uses this merit function, provided the QP subproblem has a solution, which requires bounds on the derivatives and Hessian condition number.

It is also necessary to define the penalty weights $\boldsymbol{\Theta}$ and $\boldsymbol{\Xi}$. In [59], it is shown that convergence of the method assumes the weights are chosen such that

$$M_0' \leq -\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}, \tag{2.34}$$

where M_0' denotes the directional derivative of the merit function (2.25) with respect to the steplength α evaluated at $\alpha = 0$. To achieve this, let us define the vector

$$\Psi_i = \begin{cases} \theta_i - \psi_0 & \text{if} \quad 1 \leq i \leq m, \\ \xi_{i-m} - \psi_0 & \text{if} \quad m < i \leq (m+n), \end{cases}$$

where $\psi_0 > 0$ is a strictly positive "threshold." Since (2.34) provides a single condition for the $(m+n)$ penalty parameters, we make the choice unique by minimizing the norm $\|\boldsymbol{\Psi}\|_2$. This yields

$$\boldsymbol{\Psi} = \mathbf{a}(\mathbf{a}^{\mathsf{T}}\mathbf{a})^{-1}\varsigma, \tag{2.35}$$

where

$$a_i = \begin{cases} (c_i - s_i)^2 & \text{if} \quad 1 \le i \le m, \\ (x_{i-m} - t_{i-m})^2 & \text{if} \quad m < i \le (m+n), \end{cases}$$

and

$$\begin{aligned} \varsigma \;=\; & -\frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p} + \widehat{\boldsymbol{\lambda}}^{\mathsf{T}}\Delta\mathbf{s} + \widehat{\boldsymbol{\nu}}^{\mathsf{T}}\Delta\mathbf{t} - 2(\boldsymbol{\Delta\lambda})^{\mathsf{T}}(\mathbf{c}-\mathbf{s}) - 2(\boldsymbol{\Delta\nu})^{\mathsf{T}}(\mathbf{x}-\mathbf{t}) \\ & -\psi_0(\mathbf{c}-\mathbf{s})^{\mathsf{T}}(\mathbf{c}-\mathbf{s}) - \psi_0(\mathbf{x}-\mathbf{t})^{\mathsf{T}}(\mathbf{x}-\mathbf{t}). \end{aligned}$$

Typically, the threshold parameter ψ_0 is set to machine precision and only increased if the minimum norm solution is zero. In essence, then, the penalty weights are chosen to be as small as possible consistent with the descent condition (2.34).

2.5 Hessian Approximation

A positive definite Hessian matrix ensures that the solution to the QP subproblem is unique and also makes it possible to compute $\boldsymbol{\Theta}$ and $\boldsymbol{\Xi}$ to satisfy the descent condition (2.34). For NLP applications, the Hessian of the Lagrangian is

$$\mathbf{H}_L = \nabla_x^2 F - \sum_{i=1}^{m} \lambda_i \nabla_x^2 c_i. \tag{2.36}$$

An approximation to $\mathbf{H}_L$ can be constructed using the methods described in Section 2.2; however, in general, it is not positive definite. (This does not imply that a finite difference approximation is poor, since the true Hessian may also be indefinite.) In fact, it is only necessary that the reduced Hessian of the Lagrangian be positive definite at the solution with the correct active set of constraints. Similar restrictions are required at $\mathbf{x} \neq \mathbf{x}^*$ to ensure that *each* QP subproblem has a solution. Consequently, for the QP subproblem, we use the modified matrix

$$\mathbf{H} = \mathbf{H}_L + \tau(|\sigma| + 1)\mathbf{I}. \tag{2.37}$$

The parameter τ is chosen such that $0 \le \tau \le 1$ and is normalized using the *Gerschgorin bound* for the most negative eigenvalue of $\mathbf{H}_L$, i.e.,

$$\sigma = \min_{1 \le i \le n} \left\{ h_{ii} - \sum_{i \neq j}^{n} |h_{ij}| \right\}. \tag{2.38}$$

h_{ij} is used to denote the nonzero elements of $\mathbf{H}_L$. An approach for modifying an approximation to the Hessian for least squares problems by the matrix $\bar{\tau}\mathbf{I}$ was originally suggested by Levenberg [83] and, because of this similarity, we refer to τ as the *Levenberg parameter*. As a practical matter, normalization, using the Gerschgorin bound, is useful even though the accuracy of the Gerschgorin estimate is not critical. It is also instructive to recall the trust-region interpretation of the parameter as defined by (1.93).

The proper choice for the Levenberg parameter τ can greatly affect the performance of the NLP algorithm. A fast rate of convergence can only be obtained when $\tau = 0$ and the correct active set has been identified. On the other hand, if $\tau = 1$, in order to guarantee a positive definite Hessian, the search direction $\mathbf{p}$ is significantly biased

toward a gradient direction and convergence is degraded. A strategy similar to that used for adjusting a trust region (cf. [53]) is employed by the algorithm to maintain a current value for the Levenberg parameter τ and adjust it from iteration to iteration. The *inertia* (i.e., the number of positive, negative, and zero eigenvalues) of the related KKT matrix (2.19) is used to infer that the reduced Hessian is positive definite. Using results from Gould [66], Gill et al. [61, 62] show that the reduced Hessian will be positive definite if the inertia of $\mathbf{K}_0$ (2.19) is

$$In(\mathbf{K}_0) = (n_f, m, 0), \tag{2.39}$$

where n_f is the number of rows in $\tilde{\mathbf{H}}_f$ and m is the number of rows in $\tilde{\mathbf{G}}_f$. Basically, the philosophy is to reduce the Levenberg parameter when the predicted reduction in the merit function agrees with the actual reduction, and increase the parameter when the agreement is poor. The process is accelerated by making the change in τ proportional to the observed rate of change in the gradient of the Lagrangian. To be more precise, at iteration k, three quantities are computed, namely.

1. the actual reduction

$$\varrho_1 = M^{(k-1)} - M^{(k)}, \tag{2.40}$$

2. the predicted reduction

$$\varrho_2 = M^{(k-1)} - \tilde{M}^{(k)} = -M'_0 - \frac{1}{2}\mathbf{p}^{\mathsf{T}}\mathbf{H}\mathbf{p}, \tag{2.41}$$

   where $\tilde{M}^{(k)}$ is the predicted value of the merit function, and

3. the rate of change in the norm of the gradient of the Lagrangian

$$\varrho_3 = \frac{\|\vartheta^{(k)}\|_\infty}{\|\vartheta^{(k-1)}\|_\infty}, \tag{2.42}$$

   where the error in the gradient of the Lagrangian is

$$\vartheta = \mathbf{g} - \mathbf{G}^{\mathsf{T}}\boldsymbol{\lambda} - \boldsymbol{\nu}. \tag{2.43}$$

Then, if $\varrho_1 \leq 0.25\varrho_2$, the actual behavior is much worse than predicted so bias the step toward the gradient by setting $\tau^{(k+1)} = \min(2\tau^{(k)}, 1)$. On the other hand, if $\varrho_1 \geq 0.75\varrho_2$, then the actual behavior is sufficiently close to predicted, so bias the step toward a Newton direction by setting $\tau^{(k+1)} = \tau^{(k)} \min(0.5, \varrho_3)$. It is important to note that this strategy does not ensure that the reduced Hessian is positive definite. In fact, it may be necessary to supersede this adaptive adjustment and increase $\tau^{(k+1)}$ whenever the inertia of the KKT matrix is incorrect. The inertia is easily computed as a byproduct of the symmetric indefinite factorization by counting the number of positive and negative elements in the diagonal matrix (with a positive and negative contribution coming from each 2×2 block).

2.6 Sparse SQP Algorithm

2.6.1 Minimization Process

Let us now summarize the steps in the algorithm. The iteration begins at the point $\mathbf{x}$, with $k = 1$, and proceeds as follows:

1. *Gradient Evaluation.* Evaluate gradient information $\mathbf{g}$ and $\mathbf{G}$ and then
   (a) evaluate the error in the gradient of the Lagrangian from (2.43);
   (b) terminate if the KKT conditions are satisfied;
   (c) compute $\mathbf{H}_L$ from (2.36); if this is the first iteration go to step 2; otherwise
   (d) *Levenberg modification:*
      i. compute the rate of change in the norm of the gradient of the Lagrangian from (2.42);
      ii. if $\varrho_1 \leq 0.25\varrho_2$, then set $\tau^{(k)} = \min(2\tau^{(k-1)}, 1)$; otherwise
      iii. if $\varrho_1 \geq 0.75\varrho_2$, then set $\tau^{(k)} = \tau^{(k-1)} \min(0.5, \varrho_3)$.

2. *Search Direction.* Construct the optimization search direction:
   (a) compute $\mathbf{H}$ from (2.37);
   (b) compute $\mathbf{p}$ by solving the QP subproblem (2.12)–(2.13);
   (c) *Inertia control:* if inertia of $\mathbf{K}$ is incorrect and
      i. if $\tau^{(k)} < 1$, then set $\tau^{(k)} \leftarrow \min(10\tau^{(k)}, 1)$ and return to step 2a;
      ii. if $\tau^{(k)} = 1$ and $\mathbf{H} \neq \mathbf{I}$, then set $\tau^{(k)} = 0$ and $\mathbf{H} = \mathbf{I}$ and return to step 2a;
      iii. if $\mathbf{H} = \mathbf{I}$, then QP constraints are locally inconsistent—terminate the minimization process and attempt to locate a feasible point for the nonlinear constraints;
   (d) compute $\Delta\boldsymbol{\lambda}$ and $\Delta\boldsymbol{\nu}$ from (2.29) and (2.30);
   (e) compute $\Delta\mathbf{s}$ and $\Delta\mathbf{t}$ from (2.32) and (2.33);
   (f) compute penalty parameters to satisfy (2.34); and
   (g) initialize $\alpha = 1$.

3. *Prediction.*
   (a) Compute the predicted point for the variables, the multipliers, and the slacks from (2.28);
   (b) evaluate the constraints $\bar{\mathbf{c}} = \mathbf{c}(\bar{\mathbf{x}})$ at the predicted point.

4. *Line Search.* Evaluate the merit function $M(\bar{\mathbf{x}}, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\nu}}, \bar{\mathbf{s}}, \bar{\mathbf{t}}) = \bar{M}$ and
   (a) if the merit function $\bar{M}$ is "sufficiently" less than M, then $\bar{\mathbf{x}}$ is an improved point—terminate the line search and go to step 5;
   (b) else change the steplength α to reduce M and return to step 3.

5. *Update.* Update all quantities, set $k = k + 1$;
   (a) compute the actual reduction from (2.40);
   (b) compute the predicted reduction from (2.41), where $\tilde{M}^{(k)}$ is the predicted value of the merit function; and
   (c) return to step 1.

The steps outlined describe the fundamental elements of the optimization process however, a number of points deserve additional clarification. First, note that the algorithm requires a line search in the direction defined by (2.28) with the steplength α adjusted to reduce the merit function. Adjusting the value of the steplength α, as required in step 4b, is accomplished using a line-search procedure that constructs a quadratic and cubic model of the merit function. The reduction is considered "sufficient" when $M(\alpha) - M(0) < \kappa_1 \alpha M'(0)$. Instead of (1.89), the Wolfe rule, $M'(\alpha) < \kappa_2 M'(0)$ for $0 < \kappa_1 < \kappa_2 < 1$, is imposed to prevent steplengths from becoming too small.

In order to evaluate the Hessian matrix (2.36), an estimate of the Lagrange multipliers is needed. The values obtained by solving the QP problem with $\mathbf{H} = \mathbf{I}$ are used for the first iteration and, thereafter, the values $\overline{\boldsymbol{\lambda}}$ from (2.28) are used. Note that, at the very first iteration, *two* QP subproblems are solved—one to compute first-order multiplier estimates and the second to compute the step. Furthermore, for the very first iteration the multiplier search directions are $\Delta\boldsymbol{\lambda} = 0$ and $\Delta\boldsymbol{\nu} = 0$, so that the multipliers will be initialized to the QP estimates $\overline{\boldsymbol{\lambda}} = \boldsymbol{\lambda} = \widehat{\boldsymbol{\lambda}}$ and $\overline{\boldsymbol{\nu}} = \boldsymbol{\nu} = \widehat{\boldsymbol{\nu}}$. The multipliers are reset in a similar fashion, after a *defective QP subproblem* is encountered, in step 2(c)iii. The subject of defective subproblems will be covered in Section 2.7. The Levenberg parameter τ in (2.37) and the penalty weights θ_i and ξ_i in (2.25) are initialized to zero and, consequently, the merit function is initially just the Lagrangian.

2.6.2 Algorithm Strategy

The basic algorithm described above has been implemented in FORTRAN as part of the SOCS library and is documented in [18]. In the software, the preceding approach is referred to as strategy M since the iterates follow a path from the initial point to the solution. However, in practice it may be desirable and/or more efficient to first locate a feasible point. Consequently, the software provides four different algorithm strategies namely:

M Minimize. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs until the solution $\mathbf{x}^*$ is found.

FM Find a Feasible point and then Minimize. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\mathbf{x}^f$ and then, beginning from $\mathbf{x}^f$, solve a sequence of quadratic programs until the solution $\mathbf{x}^*$ is found.

FME Find a Feasible point and then Minimize subject to Equalities. Beginning at $\mathbf{x}^0$ solve a sequence of quadratic programs to locate a feasible point $\mathbf{x}^f$ and then, beginning from $\mathbf{x}^f$, solve a sequence of quadratic programs while maintaining feasible equalities until the solution $\mathbf{x}^*$ is found.

F Find a Feasible point. Beginning at $\mathbf{x}^0$, solve a sequence of quadratic programs to locate a feasible point $\mathbf{x}^f$.

The default strategy in the software is FM since computational experience suggests that it is more robust and efficient. Details on the FME strategy can be found in [20]. The fourth strategy, F, to locate a feasible point only, is also useful when debugging a new problem formulation.

2.7 Defective Subproblems

The fundamental step in the optimization algorithm requires the solution of a QP subproblem. In Section 1.14, we discussed a number of things that can go wrong that will prevent the solution of the subproblem. In particular, the QP subproblem can be *defective* because

1. the linear constraints (2.13) are inconsistent (i.e., have no solution);
2. the Jacobian matrix $\mathbf{G}$ is rank deficient;
3. the linear constraints (2.13) are redundant or extraneous, which can correspond to Lagrange multipliers that are zero at the solution;
4. the quadratic objective (2.12) is unbounded in the null space of the active constraints.

Unfortunately, because the QP is a subproblem within the overall NLP, it is not always obvious how to determine the cause of the difficulty. In particular, the QP subproblem may be defective *locally* simply because the quadratic/linear model does not approximate the nonlinear behavior. On the other hand, the QP subproblem may be defective because the original NLP problem is inherently ill-posed. Regardless of the cause of the defective QP subproblem, the overall algorithm behavior can be significantly impacted. In particular, a defective QP subproblem often produces a large increase in the solution time because

1. there is a large amount of *fill* caused by pivoting for stability in the sparse linear algebra software, which makes the sparse method act like a dense method;
2. there are many iterations in the QP subproblem, which, in turn, necessitates the factorization of a large, dense Schur-complement matrix and/or repeated KKT sparse matrix factorizations.

In order to avoid these detrimental effects, the strategy employed by the NLP algorithm has been constructed to minimize the impact of a defective QP subproblem. The first premise in designing the NLP strategy is to segregate difficulties caused by the constraints from difficulties caused by the objective function. The second basic premise is to design an NLP strategy that will eliminate a defective subproblem rather than solve an ill-conditioned system. This philosophy has been implemented in the default FM strategy. Specifically, we find a feasible point first. During this phase, difficulties that could be attributed to the objective function, Lagrange multipliers, and Hessian matrix are ignored. Instead, difficulties are attributed solely to the constraints during this phase. After a feasible point has been located with a full-rank Jacobian, it is assumed that the constraints are OK. Thus, during the optimization phase, defects related to the objective function are treated. The primary mechanism for dealing with a defective QP subproblem during the optimization process was the Levenberg Hessian modification technique. In particular, note that in step 2(c)i of the algorithm, first the Levenberg parameter is increased. If that fails, in step 2(c)ii, the Hessian is reset to the identity matrix. Only after the above two steps fail is it concluded that the defect in the QP subproblem must be caused by the constraints, and an attempt is made to locate a (nearby) feasible point.

It is worth noting that a defective subproblem is not something unique to SQP methods. In fact, all of the globalization strategies discussed in Section 1.11 can be considered

remedies for a defective subproblem. However, it is curious that the remedies we will discuss are computationally attractive for large, sparse problems, but are generally not used for small, dense applications!

2.8 Feasible Point Strategy

2.8.1 QP Subproblem

Finding a point that is feasible with respect to the constraints is the first phase in either the FM or FME strategy and is often used when attempting to deal with a defective problem. The approach employed is to take a series of steps of the form given by (1.87) with the search direction computed to solve a *least distance program* (LDP). This can be accomplished if we impose the requirement that the search direction have minimum norm, i.e., $\|\mathbf{p}\|_2$. The *primary* method for computing the search direction is to minimize

$$\frac{1}{2}\mathbf{p}^T\mathbf{p} \tag{2.44}$$

subject to the linear constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U. \tag{2.45}$$

For problems with no inequality constraints, the LDP search direction is equivalent to

$$\mathbf{p} = -\mathbf{G}^{\#}\mathbf{b}. \tag{2.46}$$

where $\mathbf{G}^{\#}$ is the *pseudoinverse* of $\mathbf{G}$ and can be viewed as a generalization of the basic Newton step (1.28).

However, as previously suggested, it is possible to encounter a defective subproblem and, in this case, it is useful to construct the search direction from a problem that has a solution even if the Jacobian is singular and/or the linear constraints are inconsistent. Thus, the *relaxation* method requires finding the augmented set of variables $(\mathbf{p}, \mathbf{u})$ to minimize

$$\frac{1}{2}\mathbf{p}^T\mathbf{p} + \frac{\rho}{2}\mathbf{u}^T\mathbf{u} \tag{2.47}$$

subject to the linear constraints

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G}\mathbf{p} + \mathbf{u} \\ \mathbf{p} \end{bmatrix} \leq \mathbf{b}_U, \tag{2.48}$$

where the constant $\rho \gg 0$. Typically, $\rho = 10^6$. Notice that, by adding the residual or slack variables $\mathbf{u}$, the linear equations (2.48) always have a solution. Although the size of the QP problem has been increased, both the Hessian and Jacobian for the augmented problem are sparse. Although the condition number of the KKT matrix for the relaxation QP problem is $\mathcal{O}(\rho^2)$, an accurate solution can often be obtained using a few steps of iterative refinement. It is interesting that the notion of adding variables is *not* usually considered attractive for dense problems because the associated linear algebra cost becomes excessive. However, when sparse matrix techniques are employed, this technique is, in fact, quite reasonable.

Since the solution of this subproblem is based on a linear model of the constraint functions, it may be necessary to adjust the steplength α in (1.87) to produce a reduction in the constraint error. Specifically, a line search is used to adjust α to achieve "sufficient decrease" (1.89) in the constraint violation merit function as defined by

$$M_v(\mathbf{x}) = \sum_{i=1}^{m} \chi^2 (c_{Li}, c_i, c_{Ui}) + \sum_{i=1}^{n} \chi^2 (x_{Li}, x_i, x_{Ui}) \tag{2.49}$$

with $\chi(l, y, u) \equiv \max[0, l - y, y - u]$.

2.8.2 Feasible Point Strategy

The overall strategy for locating a feasible point can now be described. Beginning at the point $\mathbf{x}$, with the "primary strategy," the procedure is as follows:

1. *Gradient Evaluation.* Evaluate the constraints and Jacobian. Terminate if the constraints are feasible.

2. *Search Direction.* Compute search direction:

   (a) if *(primary strategy)*, solve the QP subproblem (2.44)–(2.45), and

      i. if QP solution is possible, then go to step 3, otherwise

      ii. change to *relaxation strategy* and go to step 2b;

   (b) if *(relaxation strategy)*, solve the QP subproblem (2.47)–(2.48).

3. *Line Search.* Choose the steplength α to reduce the constraint violation $M_v(\overline{\mathbf{x}})$ given by (2.49). If relaxation strategy is used, do an accurate line search.

4. *Strategy Modification.*

   (a) If the primary strategy is being used, then return to step 1;

   (b) if the relaxation strategy is being used and if $\alpha = 1$, then switch to primary strategy and return to step 1.

This overall algorithm gives priority to the primary method for computing the search direction. If the primary strategy fails, then presumably there is something defective with the QP subproblem and the relaxation strategy is used. If a switch to the relaxation strategy is made, then subsequent steps use this approach and perform an accurate line search. When full-length steps are taken with the relaxation strategy (i.e., $\alpha = 1$), the primary strategy is again invoked. This logic is motivated by the fact that the relaxation step with $\alpha = 1$ is "approximately" a Newton step and, therefore, it is worthwhile to switch back to the primary (LDP) step. Although this strategy is somewhat ad hoc, it has been quite effective in practice.

Detecting failure of the primary strategy in step 2a is based on a number of factors. Specifically, the primary strategy is abandoned whenever

1. the amount of "fill" in the sparse linear system exceeds expectations (implying an ill-conditioned linear system), or

2. the condition number of the KKT matrix is large, or

3. the inertia of the KKT matrix is incorrect, or

4. the number of QP iterations is excessive.

2.8.3 An Illustration

Example 2.1. The performance of the feasible point algorithm is illustrated on an example derived from an ill-conditioned two-point boundary value problem (BVP) (Burgers equation). The basic problem is to solve

$$\begin{aligned} \dot{y}_1 &= y_2, \\ \dot{y}_2 &= \epsilon^{-1} y_1 y_2, \\ 0 &\leq y_1(t) \end{aligned}$$

subject to the boundary conditions $y_1(0) = 2\tanh(\epsilon^{-1})$ and $y_1(1) = 0$. For this illustration, the parameter $\epsilon = 10^{-3}$. The continuous problem is replaced by a discrete approximation with $M = 50$ grid points. Thus, it is required to compute the values of $\mathbf{x}^T = (y_1(0), y_2(0), \ldots, y_1(1), y_2(1))$ such that the constraints

$$\mathbf{c}(\mathbf{x}) = \mathbf{y}_{j+1} - \mathbf{y}_j - \frac{h}{2}\left[\dot{\mathbf{y}}_{j+1} + \dot{\mathbf{y}}_j\right] = 0$$

for $j = 1, \ldots, (M-1)$ are satisfied in addition to the boundary conditions. For this example, the iterations began with a linear initial guess between the boundary conditions

$$\mathbf{y}_k = \mathbf{y}(0) + \frac{(k-1)}{(M-1)}\left[\mathbf{y}(1) - \mathbf{y}(0)\right]$$

for $k = 1, \ldots, M$ with $y_2(0) = -2\epsilon^{-1}[1 - \tanh(\epsilon^{-1})]$ and $y_2(1) = -2\epsilon^{-1}$. We defer details of the discretization process to subsequent chapters and simply view this as a system of nonlinear equations to be solved by proper choice of the variables $\mathbf{x}$.

Iter.	Method	KKT Cond.	α	$\|\mathbf{c}\|$
1	ldp:r	$0.48 \times 10^{+12}$	1.000	97.3976
2	ldp:r	$0.54 \times 10^{+11}$	0.317	1.83373
3	r	$0.26 \times 10^{+13}$	0.149	1.31122
4	r	$0.38 \times 10^{+13}$	0.149	1.16142
5	r	$0.31 \times 10^{+13}$	1.000	1.02214
6	ldp:r	$0.22 \times 10^{+12}$	0.46×10^{-1}	0.337310
7	r	$0.16 \times 10^{+13}$	0.35×10^{-1}	0.323168
8	r	$0.16 \times 10^{+13}$	1.000	0.308115
9	ldp	$0.14 \times 10^{+09}$	0.37×10^{-2}	0.182173
10	ldp	$0.27 \times 10^{+08}$	0.10×10^{-1}	0.181395
11	ldp	$0.50 \times 10^{+06}$	0.88×10^{-1}	0.179771
12	ldp	$0.15 \times 10^{+06}$	0.457	0.165503
13	ldp	$0.78 \times 10^{+05}$	1.000	0.892×10^{-1}
14	ldp	$0.69 \times 10^{+05}$	1.000	0.251×10^{-1}
15	ldp	$0.69 \times 10^{+05}$	1.000	0.748×10^{-4}
16	ldp	$0.69 \times 10^{+05}$	1.000	0.212×10^{-8}

Table 2.1: Burgers equation example.

The behavior of the algorithm is summarized in Table 2.1. At the first iteration, an attempt to compute the search direction using the primary least distance programming

method (ldp) failed, and the relaxation (r) strategy was used. A step of length $\alpha = 1$ reduced the constraint error $\|\mathbf{c}\|$ from 97.3976 to 1.83373. Since a full Newton step was used, the second iteration began with the primary ldp strategy, which also failed, forcing the use of the relaxation method. For the second iteration, the steplength α was 0.317 (which is not a Newton step) and, consequently, the relaxation strategy was employed for iteration 3. At iteration 6, an attempt was made to switch back to the primary strategy but it again was unsuccessful. Finally, at iteration 9, it was possible to return to the primary strategy, which was then used for all subsequent iterations. Notice that the condition number of the symmetric indefinite KKT system is rather moderate at the solution, even though it is very large for some of the early iterations.

2.9 Computational Experience

2.9.1 Large, Sparse Test Problems

The NLP algorithm described has been tested on problems derived from optimal control and data-fitting applications. An extensive collection of test results for trajectory optimization and optimal control problems is found in [24]. The test set includes simple quadratic programs, nonlinear root solving, and poorly posed problems. Also included are examples with a wide range in the number of degrees of freedom and function nonlinearity. All problems in the test set

1. exhibit a *banded* sparsity pattern for the Jacobian and Hessian and
2. have some active constraints (i.e., no unconstrained problems).

With regard to the first attribute, all problems have some nonzero elements that are not along the diagonal (i.e., they are not separable). On the other hand, none of the problems are characterized by matrices with truly random structure. In general, the results described in [19] are very positive and, consequently, it seems worthwhile to understand the reasons for this promising behavior.

The calculation and treatment of the Hessian matrix are fundamental to the observed performance of the algorithm. As stated, the basic algorithm requires that the Hessian matrix $\mathbf{H}_L$ be computed from (2.36). Observe that the evaluation of $\mathbf{H}_L$ requires an estimate for both the variables and the Lagrange multipliers, i.e., $(\mathbf{x}, \boldsymbol{\lambda})$. Since the accuracy of the Hessian is affected by the accuracy of the multipliers, it seems desirable to use an NLP strategy that tends to produce "accurate" values for $\boldsymbol{\lambda}$. The default FM strategy, which first locates a feasible point and then stays "near" the constraints, presumably benefits from multiplier estimates $\boldsymbol{\lambda}$ that are more accurate near the constraints.

A summary of the results for the test problem set in [19] is given in Figure 2.1. All 109 problems were run using the three optimization strategies (M, FM, FME). The algorithm performance was measured in terms of the number of function evaluations (the number of times $f(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$ are evaluated) and the solution time. For each test problem, a first-, second-, and third-place strategy has been selected, where the first-place strategy required the smallest number of function evaluations. If a particular strategy failed to solve the problem, this was counted as a failure. It is clear from Figure 2.1 that strategy FM was in first place over 63% of the time. Furthermore, FM was either the best or second-best strategy nearly 89% of the time. Finally, notice that strategy FM solved all 109 problems (no failures). For all but 7 cases, the least number of function evaluations corresponds to the shortest solution time. Consequently, comparing strategies based on

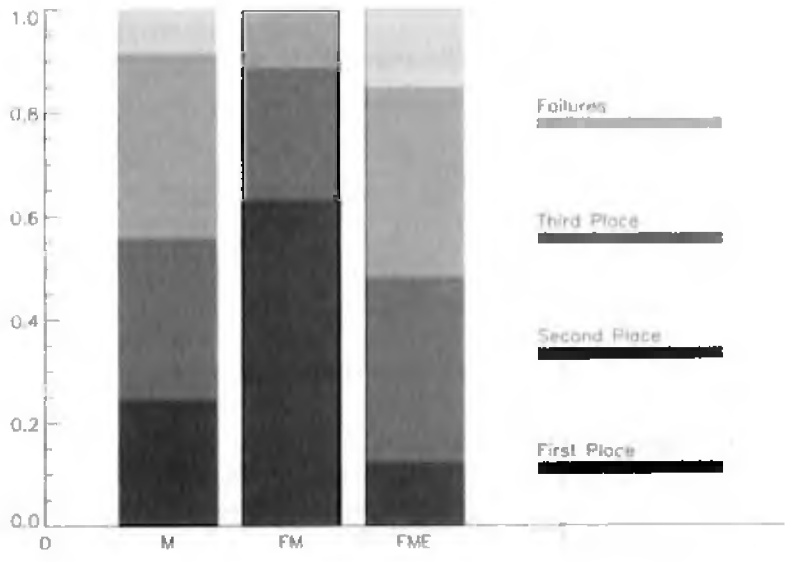


Figure 2.1: Strategy comparison.

run-time leads to the same conclusions. These results clearly indicate why strategy FM has been selected as the default. We note that for 3 problems, there are no degrees of freedom, in which case F is the only possible strategy and these cases were eliminated from the comparison.

There are at least three alternatives for computing the Hessian matrix. For some applications, it is possible to evaluate the relevant matrices analytically. Unfortunately, this is often cumbersome. For all of the test results in [19], the sparse finite difference approximations described in Section 2.2 were used. Additional information on sparse differencing is given in [22]. The third alternative, which is often effective for small, dense problems, is to use a quasi-Newton approximation to the Hessian. In general, the errors introduced by finite difference approximations are far smaller than those for quasi-Newton estimates and, for all practical purposes, one can consider the first two alternatives to be "exact." Probably the most significant advantage of methods with an exact Hessian is that ultimately one can expect quadratic convergence. The adaptive Levenberg strategy described in (2.40)–(2.43) is designed so that ultimately $\tau = 0$ and $\mathbf{H} = \mathbf{H}_L$ in the QP subproblem.

While the analysis of algorithm performance on numerical examples is always difficult because of implementation and testing issues, one can nevertheless expect to observe certain behavior in the results. When a Newton method is used to minimize an unconstrained n-dimensional quadratic function, convergence can be expected in one step, as discussed in Chapter 1. When a quasi-Newton method (initialized with $\mathbf{H} = \mathbf{I}$) is applied to a general quadratic function, convergence is expected in n steps, provided an exact line search is used. On the other hand, if the objective is not quadratic but the initial point is within the Newton region of convergence, the solution can be expected after $\kappa_1 > 1$ steps. In contrast, the quasi-Newton method can be expected to converge after $\kappa_2 n$ steps with $\kappa_2 \geq 1$. The primary advantage of a Newton versus a quasi-Newton method is that the number of iterations does not grow with the size of the problem n. For problems with constraints, one expects the behavior to be dictated by the number of

degrees of freedom, i.e., $n_d = n - \hat{m}$. The impact of quadratic convergence is dramatically illustrated by the results summarized in Table 2.2.

M	50	100	200	400	800	1600	3200	6400
n	351	701	1401	2801	5601	11201	22401	44801
m	253	503	1003	2003	4003	8003	16003	32003
$n-m$	98	198	398	798	1598	3198	6398	12798
NFE	404	448	493	404	485	394	486	620
NGE	16	17	18	16	17	15	17	20
NHE	7	8	9	7	9	7	9	12
T (sec)	4.32	9.82	22.60	46.73	209.87	580.73	2722.99	10765.0

Table 2.2: Shuttle reentry example.

The table summarizes the results for a series of eight sparse NLP problems derived from a space shuttle reentry trajectory optimization (see Example 5.2). For this example, the dynamics are described by five state variables, namely altitude, latitude, velocity, flight path angle, and azimuth angle. The shape of the trajectory is determined by choosing two control variables, the angle of attack and the bank angle. A large, sparse nonlinear program results when the continuous problem is discretized, that is, the NLP variables are values for the state and control at M grid points. The differential equations describing the motion of the vehicle are approximated by a set of nonlinear (trapezoidal discretization) constraints, which relate the quantities at the grid points. A complete description of this *direct transcription* process is given in subsequent chapters as well as in [23]. The initial guess used to begin the nonlinear program is constructed by linearly interpolating the state and control variables between the initial and final times.

The resulting NLP problem is characterized by a number of relevant properties. The Hessian matrix $\mathbf{H}_L$ has one dense row (column) and M dense (7×7) blocks along the diagonal. The Hessian matrix is well-conditioned with Gerschgorin bounds for the eigenvalues at the solution being -2.32 and $+2.96$. There are no zero Lagrange multipliers at the solution and since no modification is necessary, the reduced Hessian is strictly positive definite. The condition number of the KKT matrix (for $M = 50$) is $+5319$. The Jacobian matrix $\mathbf{G}$ has one dense column and a banded "staircase" structure with M rectangular (7×14) blocks. Although the condition number of the KKT matrix observed during the LDP steps was as large as $+626348$, no infeasible QP subproblems were encountered.

The first row of Table 2.2 defines the number of grid points for the problem, i.e., the first NLP problem had 50 grid points, the second had 100 grid points, etc. The second row presents the number of NLP variables, the third row gives the number of NLP constraints, and the number of degrees of freedom is given in row four. The largest problem has 44801 variables, 32003 nonlinear constraints, and 12798 degrees of freedom at the solution. Row five of the table (NFE) shows the total number of function evaluations (including finite difference perturbations) needed to solve the problem. Row six contains the number of Jacobian (gradient) evaluations (NGE) needed to converge, and row seven presents the number of Hessian evaluations (NHE). The final row of the table presents the cpu time (seconds) to obtain a solution on a Sun Sparcstation 20. The most striking thing about these results is that the number of function, gradient, and Hessian evaluations is nearly constant for all of the problems *regardless of size*! Since a single Jacobian evaluation is done for each QP subproblem, it is easy to infer that the smallest and largest problems were solved with nearly the same number of QP subproblems. Finally, notice that the

cpu time grows *linearly* with the size of the problem! This cost can be attributed to the fact that the computational complexity of the sparse linear algebra is $\mathcal{O}(\kappa n)$, where κ is a constant defined by the nonzero percentage of the matrix rather than $\mathcal{O}(n^3)$ for dense linear algebra.

2.9.2 Small, Dense Test Problems

Although the strategy described was developed to accommodate sparse NLP applications, it is interesting to consider whether the same techniques may also be worthwhile for small, dense problems. One significant feature developed for large, sparse problems is the Levenberg modification technique, which permits using the exact Hessian without altering the matrix sparsity. A second feature is the default FM strategy, which first locates a feasible point. In contrast to the large, sparse case, for small, dense problems it is common to use a quasi-Newton approximation for the Hessian matrix. One possible approximation is the SR1 formula given by (1.49). The update formula requires the change in position given by $\mathbf{\Delta x} = \bar{\mathbf{x}} - \mathbf{x}$. For the SR1 update, it is appropriate to use $\mathbf{\Delta g} = \nabla_x L(\bar{\mathbf{x}}) - \nabla_x L(\mathbf{x})$. If the denominator is zero, making the SR1 correction undefined, we simply skip the update. Now recall that the SR1 recursive estimate is symmetric but not necessarily positive definite. Since the reduced Hessian must be positive definite at the solution, the most common approximation is the BFGS update formula (1.50). In this case, it is possible to keep the entire approximate Hessian positive definite (thereby ensuring the reduced Hessian is positive definite) provided the update also is constructed such that $\mathbf{\Delta x}^T \mathbf{\Delta g} > 0$. We make an attempt to satisfy this condition by adjusting the Lagrange multiplier estimates used to construct the gradient difference $\mathbf{\Delta g}$. If this fails, the update is skipped. Thus, there are two alternative approaches for incorporating a recursive estimate into the NLP framework described. Since the update generated by the SR1 formula is symmetric, but indefinite, one might expect to generate a "more accurate" approximation to the Hessian $\mathbf{H}_L$. However, as in the case of an exact Hessian, it will be necessary to modify the approximation using the Levenberg strategy. In contrast, the BFGS update will not require any modification to maintain positive definiteness. Nevertheless, the BFGS approximation may not be an accurate estimate for an indefinite Hessian.

Algor.	FDH-FM	BFGS-FM	SR1-M	FDH-M	BFGS-M	NPSOL
Better	9	19	19	7	25	25
Worse	48	25	13	51	33	34
Same	3	15	28	2	7	1
Fail	1	2	1	1	4	2
Solve	0	0	1	1	0	3
Both	7	7	6	6	7	4
% Δ	70.89	22.48	2.016	69.83	22.82	16.88

Table 2.3: Dense test summary.

Table 2.3 summarizes the results of these different strategies on a set of small, dense test problems. The test set given in [69] consists of 68 test problems, nearly all of them found in the collection by Hock and Schittkowski [75]. The NLP algorithm described was used with 3 different methods to construct the Hessian. Both the FM and M strategies were employed. Finally, the NPSOL [60] algorithm was used as a benchmark. The *baseline* strategy referred to as SR1-FM incorporates the SR1 update in conjunction with the

FM option. All results in Table 2.3 are relative—thus, the second column labeled FDH-FM compares the results for a finite difference Hessian and FM option to the baseline SR1-FM performance. The number of function evaluations (including finite difference perturbations) is the quantitative measure used to assess algorithm performance. By definition, all computed quantities (objective and constraints) are evaluated on a single function evaluation. Thus, when comparing FDH-FM and SR1-FM (reading down the second column of the table), one finds that better results were obtained on 9 problems out of 68, where "better" means that the solution was obtained with fewer function evaluations. Worse results were obtained on 48 of the 68 problems using the FDH-FM option and 3 cases were the same. The FDH-FM option also failed on 1 problem that was solved by the baseline and did not solve any more problems than the baseline. Both options failed to find a solution in 7 cases. The final row in the table presents the *average percentage change* in the number of function evaluations. Thus, on average, the FDH-FM option required 70.89% more function evaluations to obtain a solution than the SR1-FM option. It should be noted that a "failure" can occur either because of an algorithmic factor (e.g., maximum iterations) or a problem characteristic (e.g., no solution exists).

An analysis of the results in Table 2.3 suggests a number of trends. First, the use of a finite difference Hessian approximation for small, dense problems is much more expensive than a recursive quasi-Newton method. Second, the SR1 update seems to be somewhat better on average than the BFGS update. Presumably this is because the SR1 update yields a better approximation to the Hessian because it does not require positive definiteness. Third, although there is a slight benefit to the FM strategy in comparison to the M strategy, the advantage is not nearly as significant as it is for large, sparse applications. One can speculate that since a recursive Hessian estimate is poor during early iterations, there is no particular advantage to having "good" multiplier estimates. Finally, it is interesting to note that the results in the last two columns comparing NPSOL with the BFGS-M strategy are quite similar, which is to be expected since they employ nearly identical methods. The minor differences in performance are undoubtedly due to different line-search algorithms and other subtle implementation issues.

2.10 Nonlinear Least Squares

2.10.1 Background

In contrast to the general NLP problem as defined by (1.98) -(1.100) in Section 1.12, the nonlinear least squares (NLS) problem is characterized by an objective function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{r}^{\mathsf{T}}(\mathbf{x})\mathbf{r}(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{\ell}\mathbf{r}_i^2, \tag{2.50}$$

where $\mathbf{r}(\mathbf{x})$ is an ℓ-vector of *residuals*. As for the general NLP, it is necessary to find an n-vector $\mathbf{x}$ to minimize $f(\mathbf{x})$ while satisfying the constraints (1.99) and bounds (1.100). The $\ell \times n$ *residual Jacobian* matrix $\mathbf{R}$ is defined by

$$\mathbf{R}^{\mathsf{T}} = [\nabla\mathbf{r}_1, \ldots, \nabla\mathbf{r}_\ell] \tag{2.51}$$

and the gradient vector is

$$\mathbf{g} = \mathbf{R}^{\mathsf{T}}\mathbf{r} = \sum_{i=1}^{\ell} r_i \nabla\mathbf{r}_i. \tag{2.52}$$

And finally, the Hessian of the Lagrangian is given by

$$\mathbf{H}_L(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{i=1}^{\ell} r_i \nabla^2 r_i - \sum_{i=1}^{m} \lambda_i \nabla^2 c_i + \mathbf{R}^T\mathbf{R} \tag{2.53}$$

$$\equiv \mathbf{V} + \mathbf{R}^T\mathbf{R}. \tag{2.54}$$

The matrix $\mathbf{R}^T\mathbf{R}$ is referred to as the *normal matrix* and we shall refer to the matrix $\mathbf{V}$ as the *residual Hessian*.

2.10.2 Sparse Least Squares

Having derived the appropriate expressions for the gradient and Hessian of the least squares objective function, one obvious approach is to simply use these quantities as required within the framework of a general NLP problem. Unfortunately, there are a number of well-known difficulties that are related to the normal matrix $\mathbf{R}^T\mathbf{R}$. First, it is quite possible that the Hessian $\mathbf{H}_L$ may be dense even when the residual Jacobian $\mathbf{R}$ is sparse. For example, this can occur when there is one dense row in $\mathbf{R}$. Even if the Hessian is not completely dense, in general, formation of the normal matrix introduces "fill" into an otherwise sparse problem. Second, it is well known that the normal matrix approach is subject to ill-conditioning even for small, dense problems. In particular, formation of $\mathbf{R}^T\mathbf{R}$ is prone to cancellation, and the condition number of $\mathbf{R}^T\mathbf{R}$ is the square of the condition number of $\mathbf{R}$. It is for this reason that the use of the normal matrix is not recommended. For small, dense least squares problems, the preferred solution technique is to introduce an orthogonal decomposition for the residual Jacobian without forming the normal matrix. Unfortunately, these techniques do not readily generalize to large, sparse systems when it is necessary to repeatedly modify the active set (and, therefore, the factorization). A survey of the various alternatives is found in [71]. In order to ameliorate the difficulties associated with the normal matrix, while maintaining the benefits of the general sparse NLP Schur-complement method, a "sparse tableau" approach has been adopted.

In general, the objective function is approximated by a quadratic model of the form

$$\mathbf{g}^T\mathbf{p} + \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p}.$$

Combining (2.12) with (2.54), let us proceed formally to "complete the square" and derive an alternative representation for the term

$$\begin{aligned}
\mathbf{p}^T\mathbf{H}\mathbf{p} &= \mathbf{p}^T\left[\mathbf{V} + \mathbf{R}^T\mathbf{R}\right]\mathbf{p} \\
&= \mathbf{p}^T\mathbf{V}\mathbf{p} + \mathbf{p}^T\mathbf{R}^T\mathbf{R}\mathbf{p} \\
&= \mathbf{p}^T\mathbf{V}\mathbf{p} + \mathbf{p}^T\mathbf{R}^T\mathbf{R}\mathbf{p} + \mathbf{p}^T\mathbf{R}^T\mathbf{R}\mathbf{p} - \mathbf{p}^T\mathbf{R}^T\mathbf{R}\mathbf{p} \\
&= \mathbf{p}^T\mathbf{V}\mathbf{p} + \mathbf{p}^T\mathbf{R}^T\mathbf{w} + \mathbf{w}^T\mathbf{R}\mathbf{p} - \mathbf{w}^T\mathbf{w} \\
&= [\mathbf{p}, \mathbf{w}]^T \begin{bmatrix} \mathbf{V} & \mathbf{R}^T \\ \mathbf{R} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{w} \end{bmatrix}.
\end{aligned} \tag{2.55}$$

Notice that in the last two lines, we have introduced ℓ new variables $\mathbf{w} = \mathbf{R}\mathbf{p}$.

As in the general NLP problem, it is necessary that the QP subproblem be well-posed. Consequently, to correct a defective QP subproblem, the *residual Hessian* is modified:

$$\overline{\mathbf{V}} = \mathbf{V} + \tau(|\sigma| + 1)\mathbf{I}, \tag{2.56}$$

where σ is the Gerschgorin bound for the most negative eigenvalue of $\mathbf{V}$. Notice that it is not necessary to modify the entire Hessian matrix for the augmented problem but simply the portion that can contribute to directions of negative curvature. However, since the artificial variables $\mathbf{w}$ are introduced, the inertia test (2.39) must become

$$In(\mathbf{K}_0) = (n_f - \ell, m + \ell, 0) \tag{2.57}$$

to account for the artificial directions.

We then solve an augmented subproblem for the variables $\mathbf{q}^\mathsf{T} = [\mathbf{p}, \mathbf{w}]^\mathsf{T}$, i.e., minimize

$$\frac{1}{2}[\mathbf{p}, \mathbf{w}]^\mathsf{T} \begin{bmatrix} \overline{\mathbf{V}} & \mathbf{R}^\mathsf{T} \\ \mathbf{R} & -\mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{w} \end{bmatrix} + \mathbf{g}^\mathsf{T}\mathbf{p} \tag{2.58}$$

subject to

$$\mathbf{b}_L \leq \begin{bmatrix} \mathbf{G} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \mathbf{w} \end{bmatrix} \leq \mathbf{b}_U. \tag{2.59}$$

An alternative form for the augmented problem is suggested in [10] and [12] that requires the explicit introduction of the constraints $\mathbf{w} = \mathbf{R}\mathbf{p}$. However, the technique defined by (2.58) and (2.59) is more compact than the approach in [10] and still avoids formation of the normal matrix. As for the general NLP problem, we choose the Levenberg parameter, $0 \leq \tau \leq 1$, such that the projected Hessian of the augmented problem is positive definite. We modify the Levenberg parameter at each iteration such that ultimately $\tau \to 0$. Observe that for linear residuals and constraints, the residual Hessian $\mathbf{V} = 0$ and, consequently, $|\sigma| = 0$. Thus, in the linear case, no modification is necessary unless $\mathbf{R}$ is rank deficient. Furthermore, for linearly constrained problems with small residuals at the solution as $\|\mathbf{r}\| \to 0$, $|\sigma| \to 0$ so the modification to the Hessian is "small" even when the Levenberg parameter $\tau \neq 0$. Finally, for large residual problems, i.e., even when $\|\mathbf{r}^*\| \neq 0$, the accelerated trust-region strategy used for the general NLP method adjusts the modification $\tau \to 0$, which ultimately leads to quadratic convergence.

2.10.3 Residual Hessian

Because NLS problems require the residual Hessian $\mathbf{V}$, a number of special techniques have been developed specifically for this purpose. One approach is to construct a quasi-Newton approximation for $\mathbf{V}$ itself rather than the full Hessian $\mathbf{H}$. Inserting the definition of the least squares Hessian (2.54) into the secant equation (1.47) gives

$$\overline{\mathbf{B}}\Delta\mathbf{x} = (\overline{\mathbf{V}} + \mathbf{R}^\mathsf{T}\mathbf{R})\Delta\mathbf{x} = \Delta\mathbf{g}. \tag{2.60}$$

Rearranging this expression leads to

$$\overline{\mathbf{V}}\Delta\mathbf{x} = \Delta\mathbf{g} - \mathbf{R}^\mathsf{T}\mathbf{R}\Delta\mathbf{x} \equiv \Delta\mathbf{g}^\#. \tag{2.61}$$

The idea of using an SR1 update (1.49) to construct a quasi-Newton approximation to $\mathbf{V}$ was proposed in [7] and [8]. Dennis, Gay, and Welsch [45] suggest a similar technique in which the DFP update (1.51) is used and the quantity

$$\boldsymbol{\Delta}\mathbf{g}^{\#} \equiv (\overline{\mathbf{R}} - \mathbf{R})^{\mathsf{T}}\overline{\mathbf{r}}. \tag{2.62}$$

Other variations have also been suggested and these methods are generally quite effective for least squares problems, especially when $\|\mathbf{r}^*\| \gg 0$.

A finite difference method can also be used to construct gradient information for NLS problems. Since both the residual Jacobian and the constraint Jacobian are needed, the quantities being differentiated in (2.1) are defined as

$$\mathbf{q} = \begin{bmatrix} \mathbf{c} \\ \mathbf{r} \end{bmatrix}, \tag{2.63}$$

and then it follows that

$$\mathbf{D} = \begin{bmatrix} \mathbf{G} \\ \mathbf{R} \end{bmatrix}. \tag{2.64}$$

It is also natural to define

$$\boldsymbol{\omega}^{\mathsf{T}} = (-\lambda_1, \ldots, -\lambda_m, r_1, \ldots, r_\ell), \tag{2.65}$$

where λ_k are the Lagrange multipliers with $v = m + \ell$, so that (2.2) becomes

$$\Omega(\mathbf{x}) = \sum_{i=1}^{v} \omega_i q_i(\mathbf{x}) = -\sum_{i=1}^{m} \lambda_i c_i(\mathbf{x}) + \sum_{i=1}^{\ell} [r_i] r_i(\mathbf{x}). \tag{2.66}$$

It should be recalled that elements of $\boldsymbol{\omega}$ are not perturbed during the finite difference operation. To emphasize this, we have written the second term above as $[r_i]r_i(\mathbf{x})$ since the quantities $[r_i]$ do *not* change during the perturbations. Then it follows that the residual Hessian $\mathbf{V} = \mathbf{E}$, where $\mathbf{E}$ is given by (2.3).

This page intentionally left blank

Chapter 3

Optimal Control Preliminaries

3.1 The Transcription Method

The preceding chapters focus on methods for solving NLP problems. In the remainder of the book, we turn our attention to the *optimal control problem.* An NLP problem is characterized by a *finite* set of variables $\mathbf{x}$ and constraints $\mathbf{c}$. In contrast, optimal control problems can involve continuous *functions* such as $\mathbf{y}(t)$ and $\mathbf{u}(t)$. It will be convenient to view the optimal control problem as an infinite-dimensional extension of an NLP problem. However, practical methods for solving optimal control problems require Newton-based iterations with a *finite* set of variables and constraints. This goal can be achieved by *transcribing* or converting the infinite-dimensional problem into a finite-dimensional approximation.

Thus, the *transcription method* has three fundamental steps:

1. convert the dynamic system into a problem with a *finite* set of variables, then

2. solve the finite-dimensional problem using a parameter optimization method (i.e., the NLP subproblem), and then

3. assess the accuracy of the finite-dimensional approximation and if necessary repeat the transcription and optimization steps.

We will begin the discussion by focusing on the first step in the process, namely identifying the NLP variables, constraints, and objective function for common applications. In simple terms, we will focus on how to convert an optimal control problem into an NLP problem.

3.2 Dynamic Systems

A *dynamic system* is usually characterized mathematically by a set of *ordinary differential equations* (ODEs). Specifically, the dynamics are described for $t_I \leq t \leq t_F$ by a system of n_y ODEs

$$\dot{\mathbf{y}} = \begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \vdots \\ \dot{y}_{n_y} \end{bmatrix} = \begin{bmatrix} f_1[y_1(t), \ldots, y_{n_y}(t), t] \\ f_2[y_1(t), \ldots, y_{n_y}(t), t] \\ \vdots \\ f_{n_y}[y_1(t), \ldots, y_{n_y}(t), t] \end{bmatrix} = \mathbf{f}[\mathbf{y}(t), t]. \tag{3.1}$$

For many applications, the variable t is time and it is common to associate the independent variable with a time scale. Of course, mathematically there is no need to make such an assumption and it is perfectly reasonable to define the independent variable in any meaningful way. The system (3.1) is referred to as an *explicit* first-order ODE system.

An important problem in differential equations is the so-called *initial value problem* (IVP). Specifically, given a set of initial values for the dependent variables $\mathbf{y}(t_I)$, called the *initial conditions*, one must determine the values at some other point t_F. In contrast, for the boundary value problem (BVP), one must determine the dependent variables such that they have specified values at two or more points, say t_I and t_F. The conditions that define the dependent variables are called *boundary conditions*. Although most optimal control problems are boundary value problems, a number of concepts are usually introduced in the initial value setting.

3.3 Shooting Method

Example 3.1. To illustrate the basic concepts, let us consider the simple dynamics:

$$\frac{dy}{dt} = y.$$

Clearly, the analytic solution to the IVP is given by

$$y = y(t_I)e^{t-t_I}.$$

However, suppose we want to find $y(t_I) \equiv y_I$ such that $y(t_F) = b$, where b is a specified value. This is called a two-point boundary value problem. In particular, using the transcription formulation, it is clear that we can formulate the problem in terms of the single (NLP) variable $x \equiv y_I$. Then it is necessary to solve the single constraint

$$\begin{aligned} c(x) &= y(t_F) - b \\ &= y_I e^{t_F - t_I} - b \\ &= x e^{t_F - t_I} - b \\ &= 0 \end{aligned}$$

by adjusting the variable x. Figure 3.1 illustrates the problem.

The approach described is referred to as the *shooting method* and is one of the simplest techniques for solving a BVP. An early practical application of the method required that a cannon be aimed such that the cannonball hit its target, hence explaining the colorful name. Of course, the method is not limited to just a single variable and constraint. In general, the shooting method can be summarized as follows:

1. guess initial conditions $\mathbf{x} = \mathbf{y}(t_I)$;
2. propagate the differential equations from t_I to t_F, i.e., "shoot";
3. evaluate the error in the boundary conditions $\mathbf{c}(\mathbf{x}) = \mathbf{y}(t_F) - \mathbf{b}$;

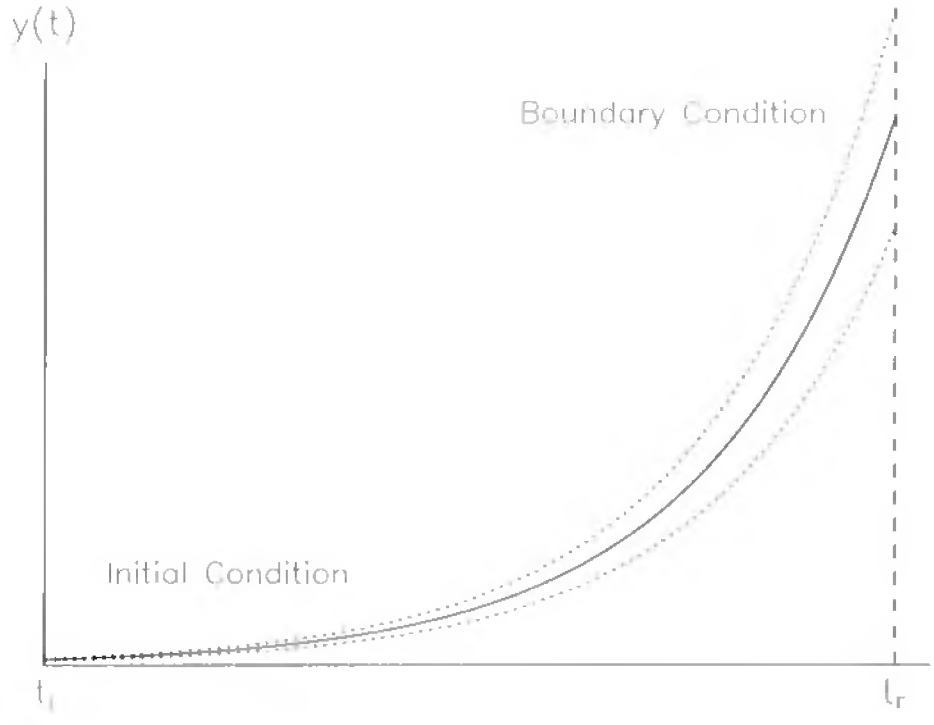


Figure 3.1: Shooting method.

4. use a nonlinear program to adjust the variables $\mathbf{x}$ to satisfy the constraints $\mathbf{c}(\mathbf{x}) = \mathbf{0}$. i.e., repeat steps 1–3.

From a practical standpoint, the shooting method is widely used primarily because the transcribed problem has a small number of variables. Clearly, the number of iteration variables is equal to the number of differential equations. Consequently, any stable implementation of Newton's method is a viable candidate as the iterative technique. Unfortunately, the shooting method also suffers from one major disadvantage. In particular, a small change in the initial condition can produce a very large change in the final conditions. This "tail wagging the dog" effect can result in constraints $\mathbf{c}(\mathbf{x})$ that are *very* nonlinear and, hence, very difficult to solve. The nonlinearity also makes it difficult to construct an accurate estimate of the Jacobian matrix that is needed for a Newton iteration. It should be emphasized that this nonlinear behavior in the boundary conditions can be caused by differential equations that are either nonlinear or *stiff* (or both). Although a discussion of stiffness is deferred temporarily, it should be clear that Example 3.1 would be much harder to solve if we replaced the *linear* differential equation $\dot{y} = y$ by the *linear* differential equation $\dot{y} = 20y$. A more realistic illustration of the shooting method is presented in Example 5.5.

3.4 Multiple Shooting Method

Example 3.2. In order to reduce the sensitivity present in a shooting method, one approach is to simply break the problem into shorter steps, i.e., don't shoot as far. Thus,

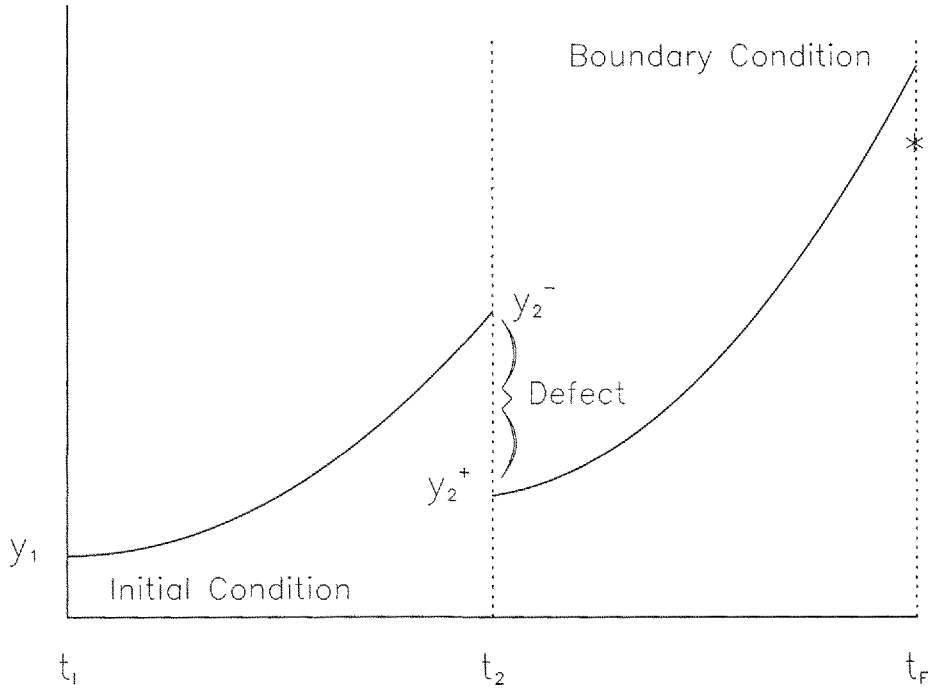


Figure 3.2: Multiple shooting method.

we might consider a "first step" for $t_I \leq t \leq t_2$ and a "second step" for $t_2 \leq t \leq t_F$. For simplicity, let us assume that $t_2 = \frac{1}{2}(t_F + t_I)$. Since the problem has now been broken into two shorter steps, we must guess a value for y at the midpoint in order to start the second step. Furthermore, we must add a constraint to force continuity between the two steps, i.e.,

$$\hat{y}_1 = y_2,$$

where $y(t_2^-) \equiv y_2^- = \hat{y}_1$ denotes the value at the end of the first step and $y_2 \equiv y(t_2^+) = y_2^+$ denotes the value at the beginning of the second step. As a result of this interval splitting, the transcribed problem now has two variables, $\mathbf{x}^\mathsf{T} \equiv [y_I, y_2]$. Furthermore, we must now solve the two constraints

$$\begin{bmatrix} c_1(\mathbf{x}) \\ c_2(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} y_2 - \hat{y}_1 \\ y(t_F) - b \end{bmatrix} = \begin{bmatrix} x_2 - x_1 e^{(t_2 - t_I)} \\ x_2 e^{(t_F - t_2)} - b \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Figure 3.2 illustrates the problem.

The approach described is called *multiple shooting* [36, 80] and the constraints enforcing continuity are called *defect* constraints or, simply, defects. Let us generalize the method as follows: compute the unknown initial values $\mathbf{y}(t_I) = \mathbf{y}_I$ such that the boundary condition

$$\mathbf{0} = \psi[\mathbf{y}(t_F), t_F] \tag{3.2}$$

holds for some value of $t_F > t_I$ that satisfies the ODE system (3.1). The fundamental idea of multiple shooting is to break the trajectory into shorter pieces or segments. Thus,

we break the time domain into smaller intervals of the form

$$t_I = t_1 < t_2 < \cdots < t_M = t_F. \tag{3.3}$$

Let us denote $\mathbf{y}_k$ for $k = 1, \ldots, (M-1)$ as the initial value for the dynamic variable at the beginning of segment k. For segment k we can propagate (integrate) the differential equations (3.1) from t_k to the end of the segment at t_{k+1}. Denote the result of this integration by $\widehat{\mathbf{y}}_k$. Collecting all segments, let us define a set of NLP variables

$$\mathbf{x}^{\mathrm{T}} = \left(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_{M-1}\right). \tag{3.4}$$

Now we also must ensure that the segments join at the boundaries: consequently, we impose the constraints

$$\mathbf{c}(\mathbf{x}) = \begin{bmatrix} \mathbf{y}_2 - \widehat{\mathbf{y}}_1 \\ \mathbf{y}_3 - \widehat{\mathbf{y}}_2 \\ \vdots \\ \boldsymbol{\psi}[\widehat{\mathbf{y}}_M, t_F] \end{bmatrix} = \mathbf{0}. \tag{3.5}$$

One obvious result of the multiple shooting approach is an increase in the size of the problem that the Newton iteration must solve. Additional variables and constraints are introduced for each shooting segment. In particular, the number of NLP variables and constraints for a multiple shooting application is $n = n_y(M-1)$, where n_y is the number of dynamic variables $\mathbf{y}$ and $(M-1)$ is the number of segments. Fortunately, the Jacobian matrix, which is needed to compute the Newton search direction, is *sparse*. In particular, only $(M-1)n_y^2$ elements in $\mathbf{G}$ are nonzero out of a possible $[(M-1)n_y]^2$. Thus, the percentage of nonzeros is proportional to $1/(M-1)$, indicating that the matrix gets sparser as the number of intervals grows. This sparsity is a direct consequence of the multiple shooting formulation, since variables early in the trajectory do not change constraints later in the trajectory. In fact, Jacobian sparsity is the mathematical consequence of *un*coupling between the multiple shooting segments. For the simple case described, the Jacobian matrix is banded with $n_y \times n_y$ blocks along the diagonal, and the very efficient methods described in Section 2.3 can be used. It is important to note that the multiple shooting segments are introduced strictly for numerical reasons. A more complete discussion of sparsity is presented in Section 4.6. Example 5.6 describes a practical application of multiple shooting to an orbit transfer problem.

An interesting benefit of the multiple shooting algorithm is the ability to exploit a parallel processor. The method is sometimes called *parallel shooting* because the simulation of each segment can be implemented on an individual processor. This technique was explored for a trajectory optimization application [21] and remains an intriguing prospect for multiple shooting methods in general.

3.5 Initial Value Problems

In the preceding sections, both the shooting and multiple shooting methods require "propagation" of a set of differential equations. For the simple motivational examples, it was possible to analytically propagate the solution from t_I to t_F (or t_i). However, in general, analytic propagation is not feasible and numerical methods must be employed. The numerical solution of the IVP for ODEs is fundamental to most optimal control

methods. The problem can be stated as follows: compute the value of $\mathbf{y}(t_F)$ for some value of $t_I < t_F$ that satisfies (3.1) with the known initial value $\mathbf{y}(t_I) = \mathbf{y}_I$. Numerical methods for solving the ODE IVP are relatively mature in comparison to the other fields in optimal control.

Most schemes can be classified as *one-step methods* or *multistep methods*. Let us begin the discussion with one-step methods. Our goal is to construct an expression over a single step from t_i to t_{i+1}, where h_i is referred to as the *integration stepsize* for step i. We denote the value of the vector $\mathbf{y}$ at t_i by $\mathbf{y}(t_i) \equiv \mathbf{y}_i$. Proceeding formally to integrate (3.1) yields

$$\begin{aligned} \mathbf{y}_{i+1} &= \mathbf{y}_i + \int_{t_i}^{t_{i+1}} \dot{\mathbf{y}} dt \\ &= \mathbf{y}_i + \int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{y}, t) dt. \end{aligned} \tag{3.6}$$

To evaluate the integral, first let us subdivide the integration step into k subintervals

$$\tau_j = t_i + h_i \rho_j \tag{3.7}$$

with

$$0 \leq \rho_1 \leq \rho_2 \leq \cdots \leq \rho_k \leq 1$$

for $1 \leq j \leq k$. With this subdivided interval, we now apply a quadrature formula within a quadrature formula. Specifically, we have

$$\int_{t_i}^{t_{i+1}} \mathbf{f}(\mathbf{y}, t) dt \approx h_i \sum_{j=1}^{k} \beta_j \widehat{\mathbf{f}}_j, \tag{3.8}$$

where $\widehat{\mathbf{f}}_j \equiv \mathbf{f}(\tau_j, \widehat{\mathbf{y}}_j)$. Notice that this approximation requires values for the variables $\mathbf{y}$ at the intermediate points τ_j, i.e., $\widehat{\mathbf{y}}(\tau_j) \equiv \widehat{\mathbf{y}}_j$. Consequently, we construct these intermediate values using the second expression

$$\int_{t_i}^{\tau_j} \mathbf{f}(\mathbf{y}, t) dt \approx h_i \sum_{\ell=1}^{k} \alpha_{j\ell} \mathbf{f}_\ell \tag{3.9}$$

for $1 \leq j \leq k$.

Collecting results, we obtain a popular family of one-step methods called the *k-stage Runge–Kutta* scheme:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \sum_{j=1}^{k} \beta_j \mathbf{f}_{ij}, \tag{3.10}$$

where

$$\mathbf{f}_{ij} = \mathbf{f}\left[\left(\mathbf{y}_i + h_i \sum_{\ell=1}^{k} \alpha_{j\ell} \mathbf{f}_{i\ell}\right), (t_i + h_i \rho_j)\right] \tag{3.11}$$

for $1 \leq j \leq k$. k is referred to as the "stage." In these expressions, $\{\rho_j, \beta_j, \alpha_{j\ell}\}$ are known constants with $0 \leq \rho_1 \leq \rho_2 \leq \cdots \leq 1$. A convenient way to define the coefficients is to use the so-called Butcher array

$$\begin{array}{c|ccc} \rho_1 & \alpha_{11} & \cdots & \alpha_{1k} \\ \vdots & \vdots & & \vdots \\ \rho_k & \alpha_{k1} & \cdots & \alpha_{kk} \\ \hline & \beta_1 & \cdots & \beta_k \end{array}.$$

The schemes are called *explicit* if $\alpha_{j\ell} = 0$ for $l \geq j$ and *implicit* otherwise. Four common examples of k-stage Runge–Kutta schemes are summarized below:

Euler Method (explicit, $k = 1$)

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}.$$

Common Representation:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}_i. \tag{3.12}$$

Classical Runge–Kutta Method (explicit, $k = 4$)

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}.$$

Common Representation:

$$\mathbf{k}_1 = h_i \mathbf{f}_i, \tag{3.13}$$

$$\mathbf{k}_2 = h_i \mathbf{f}\left(\mathbf{y}_i + \frac{1}{2}\mathbf{k}_1, t_i + \frac{h_i}{2}\right), \tag{3.14}$$

$$\mathbf{k}_3 = h_i \mathbf{f}\left(\mathbf{y}_i + \frac{1}{2}\mathbf{k}_2, t_i + \frac{h_i}{2}\right), \tag{3.15}$$

$$\mathbf{k}_4 = h_i \mathbf{f}(\mathbf{y}_i + \mathbf{k}_3, t_{i+1}), \tag{3.16}$$

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right). \tag{3.17}$$

Trapezoidal Method (implicit, $k = 2$)

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1/2 & 1/2 \\ \hline & 1/2 & 1/2 \end{array}.$$

Common Representation:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{h_i}{2}\left(\mathbf{f}_i + \mathbf{f}_{i+1}\right). \tag{3.18}$$

Hermite–Simpson Method (implicit, $k = 3$)

0	0	0	0
1/2	5/24	1/3	−1/24
1	1/6	2/3	1/6
	1/6	2/3	1/6

Common Representation:

$$\bar{\mathbf{y}} = \frac{1}{2}(\mathbf{y}_i + \mathbf{y}_{i+1}) + \frac{h_i}{8}(\mathbf{f}_i - \mathbf{f}_{i+1}), \tag{3.19}$$

$$\bar{\mathbf{f}} = \mathbf{f}_i\left(\bar{\mathbf{y}}, t_i + \frac{h_i}{2}\right), \tag{3.20}$$

$$\mathbf{y}_{i+1} = \mathbf{y}_i + \frac{h_i}{6}\left(\mathbf{f}_i + 4\bar{\mathbf{f}} + \mathbf{f}_{i+1}\right). \tag{3.21}$$

The Runge–Kutta scheme (3.10)–(3.11) is often motivated in another way. Suppose we consider approximating the solution of the ODE (3.1) by a function $\widetilde{\mathbf{y}}(t)$. As an approximation, let us use a polynomial of degree k (order $k+1$) over each step $t_i \le t \le t_{i+1}$:

$$\widetilde{\mathbf{y}}(t) = a_0 + a_1(t - t_i) + \cdots + a_k(t - t_i)^k \tag{3.22}$$

with the coefficients $(a_0, a_1, \ldots, a_k)$ chosen such that the approximation matches at the beginning of the step t_i, that is,

$$\widetilde{\mathbf{y}}(t_i) = \mathbf{y}_i, \tag{3.23}$$

and has derivatives that match at the points (3.7):

$$\frac{d\widetilde{\mathbf{y}}(\tau_j)}{dt} = \mathbf{f}[\mathbf{y}(\tau_j), \tau_j]. \tag{3.24}$$

The conditions (3.24) are called *collocation* conditions and the resulting method is referred to as a *collocation method*. Thus, the Runge–Kutta scheme (3.10)–(3.11) is a collocation method [2], and the solution produced by the method is a piecewise polynomial. While the polynomial representation (3.22) (called a monomial representation) has been introduced for simplicity in this discussion, in practice we will use an equivalent but computationally preferable form called a B-spline representation.

The collocation schemes of particular interest for the remainder of the book, namely (3.18) and (3.21), are both *Lobatto methods*. More precisely, the trapezoidal method is a Lobatto IIIA method of order 2, and the Hermite–Simpson method is a Lobatto IIIA method of order 4 [68, p. 75]. For a Lobatto method, the endpoints of the interval are also collocation points, and consequently $\rho_1 = 0$ and $\rho_k = 1$. The trapezoidal method is based on a quadratic interpolation polynomial. The three coefficients (a_0, a_1, a_2) are constructed such that the function matches at the beginning of the interval (3.23) and the slope matches at the beginning and end of the interval (3.24). For the Hermite–Simpson scheme, the four coefficients (a_0, a_1, a_2, a_3) defining a cubic interpolant are defined by matching the function at the beginning of the interval and the slope at the beginning, midpoint, and end of the interval. A unique property of Lobatto methods is that mesh points are also collocation points. In contrast, *Gauss* schemes impose the collocation

conditions strictly interior to the interval, that is, $\rho_1 > 0$ and $\rho_k < 1$. The simplest Gauss collocation scheme is the *midpoint rule*:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}\left[\frac{1}{2}(\mathbf{y}_{i+1} + \mathbf{y}_i), t_i + \frac{h_i}{2}\right]. \tag{3.25}$$

A *Radau* method imposes the collocation condition at only one end of the interval, specifically $\rho_1 > 0$ and $\rho_k = 1$. The simplest Radau collocation scheme is the *backward Euler method*:

$$\mathbf{y}_{i+1} = \mathbf{y}_i + h_i \mathbf{f}_{i+1}. \tag{3.26}$$

An obvious appeal of an explicit scheme is that the computation of each integration step can be performed without iteration; that is, given the value $\mathbf{y}_i$ at the time t_i, the value $\mathbf{y}_{i+1}$ at the new time t_{i+1} follows directly from available values of the right-hand side functions $\mathbf{f}$. In contrast, for an implicit method, the unknown value $\mathbf{y}_{i+1}$ appears nonlinearly, e.g., the trapezoidal method requires

$$0 = \mathbf{y}_{i+1} - \mathbf{y}_i - \frac{h_i}{2}\left[\mathbf{f}(\mathbf{y}_{i+1}, t_{i+1}) + \mathbf{f}(\mathbf{y}_i, t_i)\right] \equiv \boldsymbol{\zeta}_i. \tag{3.27}$$

Consequently, to compute $\mathbf{y}_{i+1}$, given the values t_{i+1}, $\mathbf{y}_i$, t_i, and $\mathbf{f}[\mathbf{y}_i, t_i]$, requires solving the nonlinear expression (3.27) to drive the *defect* $\boldsymbol{\zeta}_i$ to zero. The iterations required to solve this equation are called *corrector* iterations. An initial guess to begin the iteration is usually provided by the so-called predictor step. There is considerable latitude in the choice of predictor and corrector schemes. For some well-behaved differential equations, a single predictor and corrector step is adequate. In contrast, it may be necessary to perform multiple corrector iterations, e.g., using Newton's method, especially when the differential equations are *stiff*. To illustrate this, suppose that the dynamic behavior is described by two types of variables, namely, $\mathbf{y}(t)$ and $\mathbf{u}(t)$. Instead of (3.1), the system dynamics are described by

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \\ \epsilon \dot{\mathbf{u}} &= \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t], \end{aligned} \tag{3.28}$$

where ϵ is a "small" parameter. Within a very small region $0 \leq t \leq t_\epsilon$, the solution displays a rapidly changing behavior and, thereafter, the second equation can effectively be replaced by its limiting form[2]

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{3.29}$$

The resulting system given by

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{3.30}$$

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \tag{3.31}$$

is called a *semi-explicit differential-algebraic equation* (DAE). In modern control theory, the differential variables $\mathbf{y}(t)$ are called *state variables* and the algebraic variables $\mathbf{u}(t)$ are called *control variables*. The system is semi-explicit because the differential variables

[2]Strictly speaking $\mathbf{g}_u$ (3.37) must be uniformly negative definite.

appear explicitly (on the left-hand side), whereas the algebraic variables appear implicitly in $\mathbf{f}$ and $\mathbf{g}$. The original stiff system of ODEs (3.28) is referred to as a *singular perturbation* problem and in the limit approaches a DAE system.

The second class of integration schemes are termed *multistep methods* and have the general form

$$\mathbf{y}_{i+k} = \sum_{j=0}^{k-1} \alpha_j \mathbf{y}_{i+j} + h \sum_{j=0}^{k} \beta_j \mathbf{f}_{i+j}, \tag{3.32}$$

where α_j and β_j are known constants. If $\beta_k = 0$, then the method is explicit; otherwise, it is implicit. The *Adams schemes* are members of the multistep class that are based on approximating the functions $\mathbf{f}(t)$ by interpolating polynomials. The Adams–Bashforth method is an explicit multistep method [4], whereas the Adams–Moulton method is implicit [84]. Multistep methods must address three issues that we have not discussed for single-step methods. First, as written, the method requires information at $(k-1)$ previous points. Clearly, this implies some method must be used to start the process, and one common technique is to take one or more steps with a one-step method (e.g., Euler). Second, as written, the multistep formula assumes the stepsize h is a fixed value. When the stepsize is allowed to vary, careful implementation is necessary to ensure that the calculation of the coefficients is both efficient and well conditioned. Finally, similar remarks apply when the number of steps k (i.e., the order) of the method is changed.

Regardless of whether a one-step or multistep method is used, a successful implementation must address the accuracy of the solution. How well does the discrete solution $\mathbf{y}_i$ for $i = 1, 2, \ldots, M$, produced by the integration scheme, agree with the "real" answer $\mathbf{y}(t)$? All well-implemented schemes have some mechanism for adjusting the integration stepsize and/or order to control the integration error. The reader is urged to consult the works of Dahlquist and Björk [41], Stoer and Bulirsch [100], Hindmarsh [74], Shampine and Gordon [97], Hairer, Norsett, and Wanner [67], and Gear [56] for additional information. It is also worth noting that a great deal of discussion has been given to the distinction between explicit and implicit methods. Indeed, it is often tempting to use an explicit method simply because it is more easily implemented (and understood?). However, the optimal control problem is a boundary value problem (BVP) *not* an initial value problem, and, to quote Ascher, Mattheij, and Russell [2, p. 69],

> ... for a boundary value problem ... any scheme becomes effectively, implicit. Thus, the distinction between explicit and implicit initial value schemes becomes less important in the BVP context.

Methods for solving IVPs when dealing with a system of DAEs have appeared more recently. For a semi-explicit DAE system such as (3.30)–(3.31), it is tempting to try to "eliminate" the algebraic (control) variables in order to use a more standard method for solving ODEs. Proceeding formally to solve (3.31) one can write

$$\mathbf{u}(t) = \mathcal{G}^{-1}[\mathbf{y}, t], \tag{3.33}$$

where $\mathcal{G}^{-1}$ is used to denote the inverse of the function $\mathbf{g}$. When this value is substituted into (3.30), one obtains the nonlinear differential equation

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathcal{G}^{-1}[\mathbf{y}, t], t], \tag{3.34}$$

which is amenable to solution using any of the ODE techniques described above.

When analytic elimination is impossible, one alternative is to introduce a nonlinear iterative technique (e.g., Newton's method) that must be executed at every integration step. Consider solving the equations

$$\mathbf{0} = \mathbf{g}[\mathbf{y}_k, \mathbf{u}_k, t_k] \tag{3.35}$$

for the variables $\mathbf{u}_k$ given the variables $\mathbf{y}_k, t_k$. Applying (1.28) and (1.29), we obtain the iteration

$$\overline{\mathbf{u}}_k = \mathbf{u}_k - \mathbf{g}_u^{-1}\mathbf{g}, \tag{3.36}$$

where

$$\mathbf{g}_u = \begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \frac{\partial g_1}{\partial u_2} & \cdots & \frac{\partial g_1}{\partial u_m} \\ \frac{\partial g_2}{\partial u_1} & \frac{\partial g_2}{\partial u_2} & \cdots & \frac{\partial g_2}{\partial u_m} \\ \vdots & & \ddots & \\ \frac{\partial g_m}{\partial u_1} & \frac{\partial g_m}{\partial u_2} & \cdots & \frac{\partial g_m}{\partial u_m} \end{bmatrix}. \tag{3.37}$$

At least in principle, we could repeat the iteration (3.36) until the equations (3.35) are satisfied. This approach is not only very time-consuming, but can conflict with logic used to control integration error in the dynamic variables $\mathbf{y}$. If an implicit method is used for solving the ODEs, this "elimination" iteration must be performed within each corrector iteration; in other words, it becomes an iteration within an iteration. In simple terms, this is usually *not* the way to solve the problem! However, this discussion does provide one useful piece of information, namely, that successful application of Newton's method requires that $\mathbf{g}_u^{-1}$ can be computed, i.e., that $\mathbf{g}_u$ has full rank.

A word of caution must be interjected at this point. In order to eliminate the variables in the manner presented here, the inverse operation must exist and uniquely define the algebraic variables. However, it is not hard to construct a setting that precludes this operation. For example, suppose the matrix $\mathbf{g}_u$ is rank deficient. In this case, the algebraic constraint does not uniquely specify all of the degrees of freedom, and we could expect some subset (or subspace) of the variables to be undefined. Furthermore, it is not hard to imagine a situation where the rank deficiency in this inverse operation changes with time—full rank in some regions and rank deficient in others.

Instead of using (3.31) to eliminate the control, let us take a slightly different approach. Now if $\mathbf{g}(t) = \mathbf{0}$ must be satisfied over some range of time, then we also expect that the first derivative $\dot{\mathbf{g}} = \mathbf{0}$. Therefore, let us differentiate (3.31) with respect to t yielding

$$0 = \mathbf{g}_y\dot{\mathbf{y}} + \mathbf{g}_u\dot{\mathbf{u}} + \mathbf{g}_t \tag{3.38}$$

$$= \mathbf{g}_y\mathbf{f}[\mathbf{y}, \mathbf{u}] + \mathbf{g}_u\dot{\mathbf{u}} + \mathbf{g}_t, \tag{3.39}$$

where (3.39) follows by substituting the expression for $\dot{\mathbf{y}}$ from (3.30). Now if $\mathbf{g}_u$ is nonsingular, we can solve (3.39) for $\dot{\mathbf{u}}$ and replace the original DAE system (3.30)–(3.31) with

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \tag{3.40}$$

$$\dot{\mathbf{u}} = -\mathbf{g}_u^{-1}\left[\mathbf{g}_y\mathbf{f}[\mathbf{y}, \mathbf{u}] + \mathbf{g}_t\right]. \tag{3.41}$$

This is now just a system of differential equations in the new dynamic variables $\mathbf{z}^{\mathsf{T}} = (\mathbf{y}, \mathbf{u})$. If $\mathbf{g}_u$ is nonsingular, we say the system (3.30)- (3.31) is an *index-one DAE*. On the other hand, if $\mathbf{g}_u$ is rank deficient, we can differentiate (3.38) a second time and repeat the process. If an ODE results, we say the DAE has *index two*. It should also be clear that each differentiation may determine some (but not all) of the algebraic variables. In general, the *DAE index* is the minimum number of times that all or part of the original system must be differentiated with respect to t in order to explicitly determine the algebraic variable. A more complete definition of the DAE index can be found in Brenan, Campbell, and Petzold [30]. For obvious reasons, the process just described is called *index reduction*. It does provide one (not necessarily good) technique for solving DAEs.

The first general technique for solving DAEs was proposed by Gear [57] and uses a backward differentiation formula (BDF) in a linear multistep method. The algebraic variables $\mathbf{u}(t)$ are treated the same as the differential variables $\mathbf{y}(t)$. The method was originally proposed for the semi-explicit index-one system described by (3.30)- (3.31) and soon extended to the fully implicit form

$$\mathbf{F}[\dot{\mathbf{z}}, \mathbf{z}, t] = \mathbf{0}, \tag{3.42}$$

where $\mathbf{z} = (\mathbf{y}, \mathbf{u})$. The basic idea of the BDF approach is to replace the derivative $\dot{\mathbf{z}}$ by the derivative of the polynomial that interpolates the solution computed over the preceding k steps. The simplest example is the implicit Euler method, which replaces (3.42) with

$$\mathbf{F}\left[\frac{\mathbf{z}_i - \mathbf{z}_{i-1}}{h_i}, \mathbf{z}_i, t_i\right] = \mathbf{0}. \tag{3.43}$$

The resulting nonlinear system in the unknowns $\mathbf{z}_i$ is usually solved by some form of Newton's method at each time step t_i. The widely used production code DASSL, developed by Petzold [88, 89], essentially uses a variable-stepsize, variable-order implementation of the BDF formulas. The method is appropriate for index-one DAEs with consistent initial conditions. Current research into the solution of DAEs with higher index (≥ 2) has renewed interest in one-step methods, specifically, the implicit Runge–Kutta (IRK) schemes described for ODEs. RADAU5 [68] implements an IRK method for problems of this type. A discussion of methods for solving DAEs can be found in the books by Brenan, Campbell, and Petzold [30] and Hairer and Wanner [68].

Throughout the book, we have adopted the semi-explicit definition of a DAE as given by (3.30)–(3.31). It is worth noting that a fully implicit form such as (3.42) can be converted to the semi-explicit form by simply writing

$$\dot{\mathbf{y}} = \mathbf{u}(t), \tag{3.44}$$

$$\mathbf{0} = \mathbf{F}[\mathbf{y}, \mathbf{u}, t]. \tag{3.45}$$

However, as a rule of thumb [30], if the original implicit system (3.42) has index ν, then the semi-explicit form has index $\nu + 1$.

3.6 Boundary Value Example

Example 3.3. To motivate the boundary value problem, Alessandrini [1] describes a problem as follows:

Suppose that Arnold Palmer is on the 18th green at Pebble Beach. He needs to sink this putt to beat Jack Nicklaus and walk away with the $1,000,000 grand prize. What should he do? Solve a BVP! By modeling the surface of the green, Arnie sets up the equations of motion of his golf ball.

In [1] the acceleration acting on the golf ball is given by

$$\ddot{\mathbf{z}} = -g\mathbf{k} + gn_3\mathbf{n} - \mu_k gn_3 \frac{\dot{\mathbf{z}}}{\|\dot{\mathbf{z}}\|},$$

where the geometry of the green is modeled using a paraboloid with a minimum at $(10, 5)$ given by

$$z_3 = S(z_1, z_2) = \frac{(z_1 - 10)^2}{125} + \frac{(z_2 - 5)^2}{125} - 1, \tag{3.46}$$

where the normal force is in the direction defined by

$$\mathbf{n} = \frac{\mathbf{N}}{\|\mathbf{N}\|}, \qquad \mathbf{N} = \left(-\frac{\partial S}{\partial z_1}, -\frac{\partial S}{\partial z_2}, 1\right).$$

For this example, distance is given in feet, the constant $g = 32.174$ ft/sec^2 is the acceleration of gravity, and the constant $\mu_k = 0.2$ is called the kinetic coefficient of friction.

Now since the dynamics are described by a second-order differential equation, we can construct an equivalent first-order system by introducing new state variables $\mathbf{p} = (\mathbf{z}, \dot{\mathbf{z}})$. The dynamics are then described by the first-order system

$$\dot{p}_1 = p_4, \tag{3.47}$$

$$\dot{p}_2 = p_5, \tag{3.48}$$

$$\dot{p}_3 = p_6, \tag{3.49}$$

$$\dot{p}_4 = gn_1n_3 - \mu_k gn_3 \frac{p_4}{s}, \tag{3.50}$$

$$\dot{p}_5 = gn_2n_3 - \mu_k gn_3 \frac{p_5}{s}, \tag{3.51}$$

$$\dot{p}_6 = gn_3n_3 - \mu_k gn_3 \frac{p_6}{s} - g, \tag{3.52}$$

where $s = \sqrt{p_4^2 + p_5^2 + p_6^2}$ is the speed of the ball. The geometry has been defined such that the ball is located at $\mathbf{z} = (p_1, p_2, p_3) = (0, 0, 0)$ and the hole is located at $\mathbf{z}_H = (20, 0, 0)$. In [1], the problem is formulated as a two-point BVP with four variables $\mathbf{x} = (\dot{\mathbf{z}}(0), t_F)$ representing the initial velocity imparted when the golf club strikes the ball and the duration of the putt. Obviously, to win $1,000,000, Arnold Palmer hopes the final position of the ball is in the hole, thus producing three boundary conditions $\mathbf{z}(t_F) = \mathbf{z}_H$. Since there are four variables and only three conditions, Alessandrini suggests that the final boundary condition should be either $\|\dot{\mathbf{z}}(t_F)\| = 0$ or $\|\dot{\mathbf{z}}(t_F)\| \leq s_F$, where s_F is the maximum final speed.

Unfortunately, neither of the proposed boundary conditions is entirely satisfactory. If we use the first suggestion $\|\dot{\mathbf{z}}(t_F)\| = s = 0$, then, clearly, equations (3.50)–(3.52) have

a singularity at the solution! Obviously, this will cause difficulties when the numerical integration procedure attempts to evaluate the ODE at the final time t_F. On the other hand, suppose the second alternative is used and the integration is terminated when $0 < s \leq s_F$. This will avoid the singularity at the solution. However, the solution is not unique since there are many possible putts that will yield $s \leq s_F$ and it is not clear how to choose the value s_F.

There is a second, less obvious, difficulty with the original formulation that was pointed out by R. Vanderbei. Since the surface of the green is given by (3.46), it follows that the vertical position of the center of mass for the golf ball while on the green is just

$$p_3 = \frac{(p_1 - 10)^2}{125} + \frac{(p_2 - 5)^2}{125} - 1 + r_b, \tag{3.53}$$

where r_b is the radius of the ball. Differentiating, we find that

$$\begin{aligned} \dot{p}_3 &= \frac{2}{125}(p_1 - 10)\dot{p}_1 + \frac{2}{125}(p_2 - 5)\dot{p}_2 \\ &= \frac{2}{125}(p_1 - 10)p_4 + \frac{2}{125}(p_2 - 5)p_5 = p_6. \end{aligned} \tag{3.54}$$

When this equation is differentiated a second time, the resulting expression for $\dot{p}_6$ is not consistent with (3.52) unless the surface of the green given by (3.46) is a plane. Essentially, the formulation in [1] is "too simple" to produce a well-posed numerical problem!

A better approach is to model the motion of the golf ball in two different regions, namely, on the green and in the hole. While the ball is on the green, the motion can be defined by four state variables $\mathbf{y} = (p_1, p_2, p_4, p_5)$ and the differential equations

$$\dot{y}_1 = y_3, \tag{3.55}$$

$$\dot{y}_2 = y_4, \tag{3.56}$$

$$\dot{y}_3 = g n_1 n_3 - \mu_k g n_3 \frac{y_3}{s}, \tag{3.57}$$

$$\dot{y}_4 = g n_2 n_3 - \mu_k g n_3 \frac{y_4}{s}. \tag{3.58}$$

By using (3.54) to compute p_6, the speed of the ball $s = \sqrt{y_1^2 + y_2^2 + p_6^2(\mathbf{y})}$. One also finds that

$$\mathbf{N} = \left[-\frac{2}{125}(y_1 - 10), -\frac{2}{125}(y_2 - 5), 1 \right].$$

Thus, all of the quantities in the system (3.55)–(3.58) are completely specified by the four elements in $\mathbf{y}$. If we assume that the hole has a diameter of 4.25 in and the ball has a diameter of 1.68 in, then, when the distance from the center of the ball to the center of the hole is greater than 1.29 in, the ball is on the green. The location of the hole is just $\mathbf{y}_H = (20, 0)$. Thus, mathematically, when $\|\mathbf{y}(t) - \mathbf{y}_H\| \geq R_H$, where $R_H = (4.25/2 - 1.68/2)/12$, the ball is rolling on the green and the dynamics are given by (3.55)–(3.58).

On the other hand, when the golf ball is inside the hole, there is no frictional force term, and the surface geometry constraint is not needed. Consequently, inside the hole,

the complete dynamics are described by six (not four) state variables and

$$\dot{y}_1 = y_4, \tag{3.59}$$
$$\dot{y}_2 = y_5, \tag{3.60}$$
$$\dot{y}_3 = y_6, \tag{3.61}$$
$$\dot{y}_4 = 0, \tag{3.62}$$
$$\dot{y}_5 = 0, \tag{3.63}$$
$$\dot{y}_6 = -g. \tag{3.64}$$

Since the time when the ball leaves the green is unknown, we must introduce an additional variable t_2, where $t_2 < t_F$, and an additional constraint $\|\mathbf{y}(t_2) - \mathbf{y}_H\| = R_H$. Furthermore, as long as $\sqrt{(y_1(t_F) - 20)^2 + y_2^2(t_F)} \leq R_H$, the ball is in the hole at the end of the trajectory. However, the final state is still not uniquely defined. Although there are other possibilities, it seems reasonable to minimize the horizontal velocity at the final time, i.e., minimize $\dot{y}_1^2 + \dot{y}_2^2$. To summarize, the four NLP variables $\mathbf{x}^T = (\dot{\mathbf{y}}(0), t_2, t_F)$ must be chosen to satisfy the two NLP constraints

$$\|\mathbf{y}(t_2) - \mathbf{y}_H\| = R_H, \tag{3.65}$$
$$\sqrt{(y_1(t_F) - 20)^2 + y_2^2(t_F)} \leq R_H \tag{3.66}$$

and minimize the objective

$$F(\mathbf{x}) = \dot{y}_1^2 + \dot{y}_2^2. \tag{3.67}$$

Figure 3.3 illustrates two (local) solutions to the problem. It is interesting to note that the long-time solution takes $t_F = 4.46564$ sec with an optimal objective value $F^* = 0.1865527926$, whereas the short-time solution, which takes $t_F = 2.93612$ sec, yields an almost identical objective value of $F^* = 0.1865528358$.

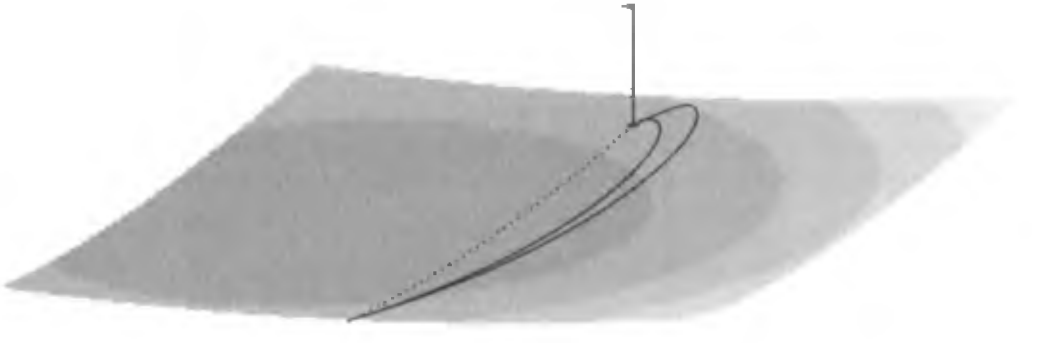


Figure 3.3: Putting example.

3.7 Dynamic Modeling Hierarchy

The golf-putting example described in the previous section suggests that it may be necessary to break a problem into one or more pieces in order to correctly model the physical

situation. In the putting example, it made sense to do this because the differential equations were different on and off the green. In fact, in general, it is convenient to view a dynamic trajectory as made up of a collection of *phases*. Specifically, let us define a *phase* as a portion of a trajectory in which the system of DAEs remains *unchanged*. Conversely, different sets of DAEs *must* be in different phases.

The complete description of a problem may require one or more phases. When necessary, phases may be *linked* together by linkage conditions. For the golf-putting example, the first phase (on the green) was linked to the second phase (in the hole) by forcing continuity in the position and velocity across the phase boundary. A phase terminates at an *event*. Thus, for the golf-putting example, phase 1 was terminated at the boundary time t_2 by the event *condition* or criterion $\|\mathbf{y}(t_2) - \mathbf{y}_H\| = R_H$. Although it is common for phases to occur sequentially in time, this is not always the case. Rather, a phase should be viewed as a fundamental building block that defines a "chunk" of the dynamics that is needed to construct a complete problem description. In fact, multiphase trajectories with nontrivial linkage conditions permit very complex problem definitions.

Now to reiterate, we have stated that

1. the DAEs must be unchanged within a phase and
2. different sets of DAEs must be in different phases.

However, this does not imply that the DAEs *must* change across a phase boundary. In other words, a phase may be introduced strictly for numerical reasons, as opposed to modeling different physical phenomena. In particular, multiple shooting segments may (or may not) be treated as phases. Furthermore, within a phase, we have a spectrum of possibilities:

1. the phase may have a single multiple shooting segment and a multiple number of integration steps;
2. the phase may be subdivided into a limited number of multiple shooting segments (e.g., 5), with multiple integration steps per segment; or
3. the number of multiple shooting segments may be *equal* to the number of integration steps (i.e., one step per segment).

Traditionally, the first approach is what we have described as the *shooting method*, and the second we have described as *multiple shooting*. Although the remainder of the book will concentrate on the third approach, it is often useful to recall this modeling hierarchy.

3.8 Function Generator

3.8.1 Description

The whole focus of this chapter has been to present different methods for transcribing a continuous problem described by DAEs into a finite-dimensional problem that can be solved using an NLP algorithm. In fact, we have described different ways to construct a *function generator* for the NLP algorithm. The input to the function generator is the set of NLP variables $\mathbf{x}$. The purpose of the function generator is to compute the constraints $\mathbf{c}(\mathbf{x})$ and objective function $F(\mathbf{x})$ by using one of the transcription methods we have outlined. If it is not possible to compute this information, the function generator

can communicate this status by setting a *function error* flag. Thus, the output of the function generator is either the requested constraint and objective function values or a flag indicating that this information cannot be computed. In fact, it is often convenient to view the function generator as a giant subroutine or "black box." Figure 3.4 illustrates this concept.

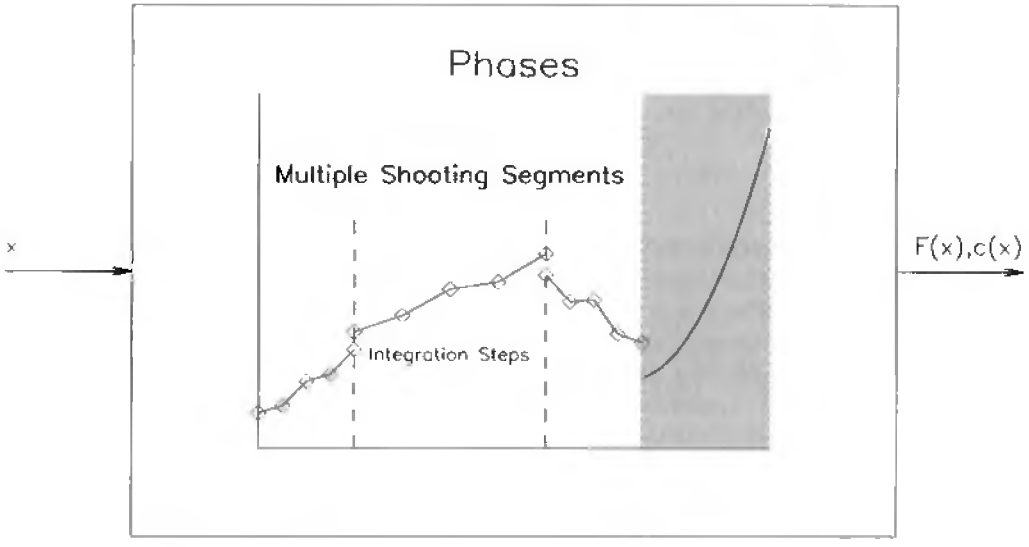


Figure 3.4: Function generator.

3.8.2 NLP Considerations

Section 1.14 described a number of considerations that should be addressed in order to construct a problem that is well-posed and amenable to solution by an NLP algorithm. How do those goals relate to the formulation of an optimal control problem? Clearly, a major goal is to construct a function generator, i.e., select a transcription method that is *noise free*. However, before proceeding any further, it is important to clarify the distinction between *consistency* and *accuracy*. To be more precise:

- A *consistent* function generator executes the same sequence of arithmetic operations for all values of $\mathbf{x}$.
- An *accurate* function generator computes accurate approximations to the dynamics $\dot{\mathbf{z}} = \mathbf{f}(\mathbf{z}, t)$.

Thus, an adaptive quadrature method, which implements a sophisticated variable-order and/or stepsize technique, will appear as "noise" to the NLP algorithm. This approach is accurate but it is not consistent! In contrast, a fixed number of steps with an explicit integration method is a consistent process, but it is not necessarily accurate.

To fully appreciate the importance of this matter, it is worth reviewing our approach. Recall that we intend to choose the NLP variables $\mathbf{x}$ to optimize an NLP objective function $F(\mathbf{x})$ and satisfy a set of NLP constraints defined by the functions $\mathbf{c}(\mathbf{x})$. Newton's method requires first and second derivative information, i.e., the Jacobian and Hessian matrices. When the NLP functions are computed by a function generator, it is clear that

we must differentiate the function generator in order to supply the requisite Jacobian and Hessian. The most common approach to computing gradients is via finite difference approximations as described in Section 2.2. A *forward difference* approximation to column j of the Jacobian matrix $\mathbf{G}$ is

$$\mathbf{G}_{.j} = \frac{1}{\delta_j} \left[\mathbf{c}(\mathbf{x} + \boldsymbol{\Delta}_j) - \mathbf{c}(\mathbf{x}) \right], \tag{3.68}$$

where the vector $\boldsymbol{\Delta}_j = \delta_j \mathbf{e}_j$ and $\mathbf{e}_j$ is a unit vector in direction j. From (2.5), a *central difference* approximation is

$$\mathbf{G}_{.j} = \frac{1}{2\delta_j} \left[\mathbf{c}(\mathbf{x} + \boldsymbol{\Delta}_j) - \mathbf{c}(\mathbf{x} - \boldsymbol{\Delta}_j) \right]. \tag{3.69}$$

In order to calculate gradient information this way, it is necessary to integrate the differential equations for each perturbation. Consequently, at least n trajectories are required to evaluate a finite difference gradient, and this information may be required for each NLP iteration. A less common alternative to finite difference gradients is to integrate the so-called variational equations. In this technique, an additional differential equation is introduced for each NLP variable and this augmented system of differential equations must be solved along with the state equations. Unfortunately, the variational equations must be derived for each application and, consequently, are used far less in general-purpose software.

While the cost of computing derivatives in this way is a matter of practical concern, a more important issue is the accuracy of the gradient information. Forward difference estimates are of order δ, whereas central difference estimates are $\mathcal{O}(\delta^2)$. Of course, the more accurate central difference estimates are twice as expensive as forward difference gradients. Typically, numerical implementations use forward difference estimates until nearly converged and then switch to the more accurate derivatives for convergence. While techniques for selecting the finite difference perturbation size might seem to be critical to accurate gradient evaluation, a number of effective methods are available to deal with this matter [63]. A more crucial matter is the interaction between the gradient computations and the underlying numerical interpolation and integration algorithms. We have already discussed how linear interpolation of tabular data can introduce gradient errors. However, it should be emphasized that sophisticated predictor-corrector variable-step, variable-order numerical integration algorithms also introduce "noise" into the gradients. Why? Because the evaluation of a perturbed trajectory is *not* consistent with the nominal trajectory. Thus, while these sophisticated techniques enhance the efficiency of the integration, they degrade the efficiency of the optimization. In fact, a simple fixed-step, fixed-order integrator may yield better overall efficiency in the optimization because the gradient information is more accurate. Two integration methods that are suitable for use inside the function generator are described in Brenan [29], Gear and Vu [58], and Vu [106]. Another issue arises in the context of an optimal control application when the final time t_F is defined implicitly by a boundary or event condition instead of explicitly. In this case, we are not asking to integrate from t_I to t_F but rather from t_I *until* $\psi[\mathbf{y}(t_F), t_F] = 0$. Most numerical integration schemes *interpolate* the solution to locate the final point. On the other hand, if the final point is found by *iteration* (e.g., using a root-finding method), the net effect is to introduce noise into the external Jacobian evaluations. A better alternative is to simply add an extra variable and constraint to the overall NLP problem and avoid the use of an "internal" iteration.

In order to avoid the aforementioned difficulties, the approach we will describe in the next chapter does two things. First, we select a transcription method with one integration step per multiple shooting segment. This ensures the function generator is *consistent*. Second, we will treat the solution accuracy *outside* the NLP problem.

This page intentionally left blank

Chapter 4

The Optimal Control Problem

4.1 Introduction

4.1.1 Dynamic Constraints

The optimal control problem may be interpreted as an extension of the NLP problem to an infinite number of variables. For fundamental background in the associated *calculus of variations*, the reader should refer to Bliss [28]. First, let us consider a simple problem with a single phase and no path constraints. Specifically, suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \phi\left[\mathbf{y}(t_F), t_F\right] \tag{4.1}$$

subject to the state equations

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{4.2}$$

and the boundary conditions

$$\boldsymbol{\psi}[\mathbf{y}(t_F), \mathbf{u}(t_F), t_F] = \mathbf{0}, \tag{4.3}$$

where the initial conditions $\mathbf{y}(t_I) = \mathbf{y}_I$ are given at the fixed initial time t_I and the final time t_F is free. This is a very simplified version (called an *autonomous form*) of the problem that will be addressed later in the chapter. We have intentionally chosen a problem with only equality constraints. However, in contrast to the discussion in Chapters 1 and 2, we now have two types of constraints. The equality constraint (4.2) may be viewed as "continuous" since it must be satisfied over the entire interval $t_I \leq t \leq t_F$, whereas the equality (4.3) may be viewed as "discrete" since it is imposed at the specific time t_F. In a manner analogous to the definition of the Lagrangian function (1.55), we form an augmented performance index

$$\hat{J} = \left[\phi + \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\psi}\right]_{t_F} - \int_{t_I}^{t_F} \boldsymbol{\lambda}^{\mathsf{T}}(t)\left\{\dot{\mathbf{y}} - \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)]\right\} dt. \tag{4.4}$$

Notice that, in addition to the Lagrange multipliers $\boldsymbol{\nu}$ for the discrete constraints, we also have multipliers $\boldsymbol{\lambda}(t)$, referred to as *adjoint or costate variables* for the continuous

(differential equation) constraints. In the finite-dimensional case, the necessary conditions for a constrained optimum (1.56) and (1.57) were obtained by setting the first derivatives of the Lagrangian to zero. In this setting, the analogous operation is to set the *first variation* $\delta\hat{J} = 0$. It is convenient to define the *Hamiltonian*

$$H = \boldsymbol{\lambda}^{\mathsf{T}}(t)\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)] \tag{4.5}$$

and the auxiliary function

$$\Phi = \phi + \boldsymbol{\nu}^{\mathsf{T}}\boldsymbol{\psi}. \tag{4.6}$$

The necessary conditions, referred to as the *Euler–Lagrange equations*, which result from setting the first variation to zero, in addition to (4.2) and (4.3), are

$$\dot{\boldsymbol{\lambda}} = -\mathbf{H_y}^{\mathsf{T}}, \tag{4.7}$$

called the *adjoint equations*,

$$\mathbf{0} = \mathbf{H_u}^{\mathsf{T}}, \tag{4.8}$$

called the *control equations*, and

$$\boldsymbol{\lambda}(t_F) = \boldsymbol{\Phi_y}^{\mathsf{T}}\Big|_{t=t_F}, \tag{4.9}$$

$$0 = (\Phi_t + H)|_{t=t_F}, \tag{4.10}$$

$$\mathbf{0} = \boldsymbol{\lambda}(t_I), \tag{4.11}$$

called the *transversality conditions*. In these expressions, the partial derivatives $\mathbf{H_y}$, $\mathbf{H_u}$, and $\boldsymbol{\Phi_y}$ are considered row vectors, i.e., $\mathbf{H_y} \equiv (\partial H/\partial y_1, \ldots, \partial H/\partial y_n)$. The control equations (4.8) are a simplified statement of the *Pontryagin maximum principle* [91]. A more general expression is

$$\mathbf{u} = \arg\min_{\mathbf{u}\in\mathcal{U}} H, \tag{4.12}$$

where $\mathcal{U}$ defines the domain of feasible controls. Note that the algebraic sign has been changed such that (4.12) is really a "minimum" principle in order to be consistent with the algebraic sign conventions used elsewhere, even though, in the original reference [91], Pontryagin derived a "maximum" principle. The maximum principle states that the control variable must be chosen to optimize the Hamiltonian (at every instant in time) subject to limitations on the control imposed by state and control path constraints. In essence, the maximum principle is a constrained optimization problem in the variables $\mathbf{u}(t)$ at all values of t. The complete set of necessary conditions consists of a DAE system (4.2), (4.7), and (4.8) with boundary conditions at both t_I and t_F (4.9), (4.10), (4.11), and (4.3). This is often referred to as a *two-point boundary value problem*. A more extensive presentation of this material can be found in Bryson and Ho [35].

4.1.2 Algebraic Equality Constraints

Generalizing the problem in the previous section, let us assume that we impose algebraic *path constraints* of the form

$$\mathbf{0} = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t] \tag{4.13}$$

in addition to the other conditions (4.2), (4.3). The treatment of this path constraint depends on the matrix of partial derivatives $\mathbf{g}_u$ (3.37). Two possibilities exist. If the matrix $\mathbf{g}_u$ is full rank, then the system of differential and algebraic equations (4.2), (4.13) is a DAE of *index one*, and (4.13) is termed a *control variable equality constraint*. For this case, the Hamiltonian (4.5) is replaced by

$$H = \boldsymbol{\lambda}^\mathsf{T}\mathbf{f} + \boldsymbol{\mu}^\mathsf{T}\mathbf{g}, \tag{4.14}$$

which will result in modification to both the adjoint equations (4.7) and the control equations (4.8).

The second possibility is that the matrix $\mathbf{g}_u$ is rank deficient. In this case, we can differentiate (4.13) with respect to t and reduce the index of the DAE as discussed in Section 3.5 (cf. (3.38)–(3.41)). The result is a new path-constraint function $\mathbf{g}'$, which is mathematically equivalent provided that the original constraint is imposed at some point on the path, say $0 = \mathbf{g}[\mathbf{y}(t_I), \mathbf{u}(t_I), t_I]$. For this new path function, again the matrix $\mathbf{g}'_u$ may be full rank or rank deficient. If the matrix is full rank, the original DAE system is said to have *index two* and this is referred to as a *state variable constraint* of order one. In the full-rank case, we may redefine the Hamiltonian using $\mathbf{g}'$ in place of $\mathbf{g}$. Of course, if the matrix $\mathbf{g}'_u$ is rank deficient, the process must be repeated. This is referred to as *index reduction* in the DAE literature [2, 30]. It is important to note that index reduction may be difficult to perform and imposition of a high-index path constraint may be prone to numerical error. This technique is illustrated in Example 5.9 for a multibody mechanism.

4.1.3 Singular Arcs

In the preceding section we addressed the DAE system

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t], \tag{4.15}$$
$$\mathbf{0} = \mathbf{g}[\mathbf{y}, \mathbf{u}, t], \tag{4.16}$$

which can appear when path constraints are imposed on the optimal control problem. However, even in the absence of path constraints, the necessary conditions (4.2), (4.7), and (4.8) lead to the DAE system

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t], \tag{4.17}$$
$$\dot{\boldsymbol{\lambda}} = -\mathbf{H_y}^\mathsf{T}, \tag{4.18}$$
$$\mathbf{0} = \mathbf{H_u}^\mathsf{T}. \tag{4.19}$$

When viewed in this fashion, the differential variables are $(\mathbf{y}, \boldsymbol{\lambda})$ and the algebraic variables are $\mathbf{u}$. Because it is a DAE system, one expects the algebraic condition to define the algebraic variables. Thus, one expects the optimality condition $\mathbf{0} = \mathbf{H_u}^\mathsf{T}$ to define the control variable provided the matrix $\mathbf{H_{uu}}$ is nonsingular. However, if $\mathbf{H_{uu}}$ is a singular matrix, the control $\mathbf{u}$ is not uniquely defined by the optimality condition. This situation is referred to as a *singular arc*, and the analysis of this problem involves techniques quite similar to those discussed above for path constraints. Furthermore, singular arc problems are not just mathematical curiosities since $\mathbf{H_{uu}}$ is singular whenever $H[\mathbf{y}, \mathbf{u}, t]$ is a linear function of $\mathbf{u}$, which can occur for linear ODEs with no path constraints. The famous sounding rocket problem proposed by Robert Goddard in 1919 [102] contains a singular arc. Recent work in periodic optimal flight [99, 95] and the analysis of wind shear during landing [6] involves formulations with singular arcs.

4.1.4 Algebraic Inequality Constraints

The preceding sections have addressed the treatment of equality path constraints. Let us now consider inequality path constraints of the form

$$\mathbf{0} \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{4.20}$$

Unlike an equality constraint, which must be satisfied for all $t_I \leq t \leq t_F$, inequality constraints may either be active ($\mathbf{0} = \mathbf{g}$) or inactive ($\mathbf{0} < \mathbf{g}$) at each instant in time. In essence, the time domain is partitioned into constrained and unconstrained subarcs. During the unconstrained arcs, the necessary conditions are given by (4.2), (4.7), and (4.8), whereas the conditions with modified Hamiltonian (4.14) are applicable in the constrained arcs. Thus, the imposition of inequality constraints presents three major complications. First, the *number* of constrained subarcs present in the optimal solution is not known a priori. Second, the *location* of the *junction points* when the transition from constrained to unconstrained (and vice versa) occurs is unknown. Finally, at the junction points, it is possible that both the control variables $\mathbf{u}$ and the adjoint variables $\boldsymbol{\lambda}$ are discontinuous. Additional *jump conditions*, which are essentially boundary conditions imposed at the junction points, must be satisfied. Thus, what was a two-point BVP may become a multipoint BVP when inequalities are imposed. For a more complete discussion of this subject, the reader is referred to the tutorial by Pesch [87] and the text by Bryson and Ho [35].

4.2 Necessary Conditions for the Discrete Problem

To conclude the discussion, let us reemphasize the relationship between optimal control and NLP problems with a simple example. Suppose we must choose the control functions $\mathbf{u}(t)$ to minimize

$$J = \phi\left[\mathbf{y}(t_F), t_F\right] \tag{4.21}$$

subject to the state equations

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t)]. \tag{4.22}$$

Let us assume that the initial and final times t_I and t_F are fixed and the initial conditions $\mathbf{y}(t_I) = \mathbf{y}_I$ are given. Let us define NLP variables

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{u}_1, \mathbf{y}_2, \mathbf{u}_2, \mathbf{y}_3, \mathbf{u}_3, \ldots, \mathbf{y}_M, \mathbf{u}_M) \tag{4.23}$$

as the values of the state and control evaluated at $t_1, t_2, \ldots, t_M$, where $t_{k+1} = t_k + h$ with $h = t_F/M$. Now

$$\dot{\mathbf{y}} \approx \frac{\mathbf{y}_{k+1} - \mathbf{y}_k}{h}. \tag{4.24}$$

Let us substitute this approximation into (4.22), thereby defining the NLP (*defect*) constraints

$$c_k(\mathbf{x}) \equiv \boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - h\mathbf{f}(\mathbf{y}_k, \mathbf{u}_k) = 0 \tag{4.25}$$

for $k = 1, \ldots, M-1$ and NLP objective function

$$F(\mathbf{x}) = \phi(\mathbf{y}_M). \tag{4.26}$$

The problem defined by (4.23), (4.25), and (4.26) is a nonlinear program. From (1.55), the Lagrangian is

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}) = F(\mathbf{x}) - \boldsymbol{\lambda}^{\mathsf{T}}\mathbf{c}(\mathbf{x}) = \phi(\mathbf{y}_M) \\ - \sum_{k=1}^{M-1} \boldsymbol{\lambda}_k^{\top} \left[\mathbf{y}_{k+1} - \mathbf{y}_k - h\mathbf{f}(\mathbf{y}_k, \mathbf{u}_k)\right]. \end{aligned} \tag{4.27}$$

The necessary conditions for this problem follow directly from the definitions (1.58) and (1.59):

$$\frac{\partial L}{\partial \boldsymbol{\lambda}_k} = \mathbf{y}_{k+1} - \mathbf{y}_k - h\mathbf{f}(\mathbf{y}_k, \mathbf{u}_k) = 0, \tag{4.28}$$

$$\frac{\partial L}{\partial \mathbf{y}_k} = (\boldsymbol{\lambda}_k - \boldsymbol{\lambda}_{k-1}) + h\boldsymbol{\lambda}_k^{\top} \frac{\partial \mathbf{f}}{\partial \mathbf{y}_k} = 0, \tag{4.29}$$

$$\frac{\partial L}{\partial \mathbf{u}_k} = h\boldsymbol{\lambda}_k^{\top} \frac{\partial \mathbf{f}}{\partial \mathbf{u}_k} = 0, \tag{4.30}$$

$$\frac{\partial L}{\partial \mathbf{y}_M} = -\boldsymbol{\lambda}_{M-1} + \frac{\partial \phi}{\partial \mathbf{y}_M} = 0. \tag{4.31}$$

Now let us consider the limiting form of this problem as $M \to \infty$ and $h \to 0$. Clearly, in the limit, equation (4.28) becomes the state equation (4.2), equation (4.29) becomes the adjoint equation (4.7), equation (4.30) becomes the control equation (4.8), and equation (4.31) becomes the transversality condition (4.9). Essentially, we have demonstrated that the Karush–Kuhn–Tucker NLP necessary conditions approach the optimal control necessary conditions as the number of variables grows. The NLP Lagrange multipliers can be interpreted as discrete approximations to the optimal control adjoint variables. While this discussion is of theoretical importance, it also suggests a number of ideas that are the basis of modern numerical methods. In particular, if the analysis is extended to inequality-constrained problems, it is apparent that the task of identifying the NLP active set is equivalent to defining constrained subarcs and junction points in the optimal control setting. Early results on this transcription process can be found in Canon, Cullum, and Polak [38], Polak [90], and Tabak and Kuo [101]. Since the most common type of transcription employs *collocation* [70], the terms "collocation" and "transcription" are often used synonymously. More recently, interest has focused on using alternative methods of discretization [50, 73, 104]. O. von Stryk [103] presents results using a Hermite approximation for the state variables and a linear control approximation, which leads to discrete approximations that are scalar multiples of those presented here.

Table 4.1 summarizes some of the major similarities between the discrete and continuous formulations of the problem.

4.3 Direct versus Indirect Methods

When describing methods for solving optimal control problems, a technique is often classified as either a *direct method* or an *indirect method*. The distinction between a direct

Discrete		Continuous	
$(\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_M)$	NLP variables	$\mathbf{y}(t)$	State variables
$(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M)$	NLP variables	$\mathbf{u}(t)$	Control variables
$\zeta_k = 0$	Defect constraints	$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t)$	State equations
$(\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_M)$	Lagrange multipliers	$\boldsymbol{\lambda}(t)$	Adjoint variables

Table 4.1: Discrete versus continuous formulation.

method and an indirect method was first introduced in Section 1.4 when considering the minimization of a univariate function. A direct method constructs a sequence of points $x_1, x_2, \ldots, x^*$ such that the objective function is minimized and typically $F(x_1) > F(x_2) > \cdots > F(x^*)$. An indirect method attempts to find a root of the necessary condition $F'(x) = 0$. At least in principle, the direct method only needs to compare values for the objective function. In contrast, the indirect method must *compute* the slope $F'(x)$ and then decide if it is sufficiently close to zero. In essence, an indirect method attempts to locate a root of the necessary conditions. A direct method attempts to find a minimum of the objective (or Lagrangian) function.

How does this categorization extend to the optimal control setting? An indirect method attempts to solve the optimal control necessary conditions (4.2), (4.3), (4.7), (4.8), (4.9), (4.10), and (4.11). Thus, for an indirect method, it is necessary to explicitly derive the adjoint equations, the control equations, and all of the transversality conditions. In contrast, a direct method does not require explicit derivation and construction of the necessary conditions. A direct method does not construct the adjoint equations (4.7), the control equations (4.8), nor any of the transversality (boundary) conditions (4.9)–(4.11). It is also important to emphasize that there is no correlation between the method used to solve the problem and the formulation. For example, one may consider applying a multiple shooting solution technique to either an indirect *or* a direct formulation. A survey of the more common approaches can be found in [13].

So *why not use the indirect method?* There are at least three major difficulties that detract from an indirect method in practice.

1. The user must compute the quantities $\mathbf{H}_\mathbf{y}$, $\mathbf{H}_\mathbf{u}$, etc., that appear in the defining equations (4.7)–(4.11). Unfortunately, this operation requires an intelligent user with at least some knowledge of optimal control theory. Furthermore, even if the user is familiar with the requisite theoretical background, it may be very difficult to construct these expressions for complicated black box applications. Finally, the approach is not flexible, since each time a new problem is posed, a new derivation of the relevant derivatives is called for!

2. If the problem description includes path inequalities (4.20), it is necessary to make an a priori estimate of the constrained-arc sequence. Unfortunately, this can be quite difficult. In fact, if the number of constrained subarcs is unknown, then the *number* of iteration variables is also unknown. Furthermore, the *sequence* of constrained/unconstrained arcs is unknown, which makes it extremely difficult to impose the correct junction conditions and, thereby, define the arc boundaries.

3. Finally, the basic method is not robust. One difficulty is that the user must guess values for the adjoint variables $\boldsymbol{\lambda}$, which, because they are not physical quantities, is very nonintuitive! Even with a reasonable guess for the adjoint variables, the numerical solution of the adjoint equations can be very ill-conditioned! The

sensitivity of the indirect method has been recognized for some time. Computational experience with the technique in the late 1960s is summarized by Bryson and Ho [35, p. 214]:

> The main difficulty with these methods is *getting started*; i.e., finding a first estimate of the unspecified conditions at one end that produces a solution reasonably close to the specified conditions at the other end. The reason for this peculiar difficulty is the extremal solutions are often *very sensitive* to small changes in the unspecified boundary conditions.... Since the system equations and the Euler–Lagrange equations are coupled together, it is not unusual for the numerical integration, with poorly guessed initial conditions, to produce "wild" trajectories in the state space. These trajectories may be so wild that values of $x(t)$ and/or $\lambda(t)$ exceed the numerical range of the computer!

Because of these practical difficulties with an indirect formulation, the remainder of the book will focus on direct methods.

4.4 General Formulation

An optimal control problem can be formulated as a collection of N *phases* as described in Section 3.7. In general, the independent variable t for *phase* k is defined in the region $t_I^{(k)} \leq t \leq t_F^{(k)}$. For many applications, the independent variable t is *time* and the phases are sequential, that is, $t_I^{(k+1)} = t_F^{(k)}$. However, neither of these assumptions is required. Within phase k, the dynamics of the system are described by a set of *dynamic* variables

$$\mathbf{z} = \begin{bmatrix} \mathbf{y}^{(k)}(t) \\ \mathbf{u}^{(k)}(t) \end{bmatrix} \tag{4.32}$$

made up of the $n_y^{(k)}$ *state variables* and the $n_u^{(k)}$ *control variables*, respectively. In addition, the dynamics may incorporate the $n_p^{(k)}$ *parameters* $\mathbf{p}^{(k)}$ that are independent of t.

Typically, the dynamics of the system are defined by a set of ODEs written in explicit form, which are referred to as the *state equations*,

$$\dot{\mathbf{y}}^{(k)} = \mathbf{f}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t], \tag{4.33}$$

where $\mathbf{y}^{(k)}$ is the $n_y^{(k)}$ dimension state vector. In addition, the solution must satisfy algebraic *path constraints* of the form

$$\mathbf{g}_L^{(k)} \leq \mathbf{g}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \leq \mathbf{g}_U^{(k)}, \tag{4.34}$$

where $\mathbf{g}^{(k)}$ is a vector of size $n_g^{(k)}$, as well as simple bounds on the state variables

$$\mathbf{y}_L^{(k)} \leq \mathbf{y}^{(k)}(t) \leq \mathbf{y}_U^{(k)} \tag{4.35}$$

and control variables

$$\mathbf{u}_L^{(k)} \leq \mathbf{u}^{(k)}(t) \leq \mathbf{u}_U^{(k)}. \tag{4.36}$$

Note that an equality constraint can be imposed if the upper and lower bounds are equal, e.g., $[g_L^{(k)}]_j = [g_U^{(k)}]_j$ for some j. This approach is consistent with our general formulation of the NLP problem as described in Section 1.12. Note that by definition, a *state variable* is a differentiated variable that appears on the left-hand side of the differential equation (4.33). In contrast, a *control variable* is an algebraic variable. Although this terminology is convenient for most purposes, there is often some ambiguity present when constructing a mathematical model of a physical system. For example, in multibody dynamics, it is common to model some physical "states" by algebraic variables.

The phases are linked by boundary conditions of the form

$$\begin{aligned}\boldsymbol{\psi}_L \le \boldsymbol{\psi} \Big[& \mathbf{y}^{(1)}(t_I^{(1)}), \mathbf{u}^{(1)}(t_I^{(1)}), \mathbf{p}^{(1)}, t_I^{(1)}, \\ & \mathbf{y}^{(1)}(t_F^{(1)}), \mathbf{u}^{(1)}(t_F^{(1)}), \mathbf{p}^{(1)}, t_F^{(1)}, \\ & \mathbf{y}^{(2)}(t_I^{(2)}), \mathbf{u}^{(2)}(t_I^{(2)}), \mathbf{p}^{(2)}, t_I^{(2)}, \\ & \mathbf{y}^{(2)}(t_F^{(2)}), \mathbf{u}^{(2)}(t_F^{(2)}), \mathbf{p}^{(2)}, t_F^{(2)}, \\ & \cdots \\ & \mathbf{y}^{(N)}(t_I^{(N)}), \mathbf{u}^{(N)}(t_I^{(N)}), \mathbf{p}^{(N)}, t_I^{(N)}, \\ & \mathbf{y}^{(N)}(t_F^{(N)}), \mathbf{u}^{(N)}(t_F^{(N)}), \mathbf{p}^{(N)}, t_F^{(N)} \Big] \le \boldsymbol{\psi}_U.\end{aligned} \tag{4.37}$$

In spite of its daunting appearance, (4.37) has a rather simple interpretation. Basically, the boundary conditions allow the value of the dynamic variables at the beginning and end of any phase to be related to each other. For example, we might have an expression of the form

$$0 = y_1^{(1)}\left[t_F^{(1)}\right] - \cos\left[y_2^{(3)}(t_I^{(3)})\right],$$

which requires the value of the first state variable at the end of phase 1 to equal the cosine of the second state at the beginning of phase 3.

Finally, it may be convenient to evaluate expressions of the form

$$\int_{t_I^{(k)}}^{t_F^{(k)}} \mathbf{w}^{(k)}\left[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t\right] dt, \tag{4.38}$$

which involve the *quadrature functions* $\mathbf{w}^{(k)}$. Collectively, we refer to those functions evaluated during the phase, namely

$$\mathbf{F}^{(k)}(t) = \begin{bmatrix} \mathbf{f}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \\ \mathbf{g}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \\ \mathbf{w}^{(k)}[\mathbf{y}^{(k)}(t), \mathbf{u}^{(k)}(t), \mathbf{p}^{(k)}, t] \end{bmatrix}, \tag{4.39}$$

as the vector of *continuous functions*. Similarly, functions evaluated at specific points, such as the boundary conditions $\boldsymbol{\psi}(\cdot)$, are referred to as *point functions*.

The basic optimal control problem is to determine the $n_u^{(k)}$-dimensional control vectors $\mathbf{u}^{(k)}(t)$ and parameters $\mathbf{p}^{(k)}$ to minimize the performance index

$$
\begin{aligned}
J = \phi \Big[& \mathbf{y}^{(1)}(t_I^{(1)}), \mathbf{u}^{(1)}(t_I^{(1)}), \mathbf{p}^{(1)}, t_I^{(1)}, \\
& \mathbf{y}^{(1)}(t_F^{(1)}), \mathbf{u}^{(1)}(t_F^{(1)}), \mathbf{p}^{(1)}, t_F^{(1)}, \\
& \mathbf{y}^{(2)}(t_I^{(2)}), \mathbf{u}^{(2)}(t_I^{(2)}), \mathbf{p}^{(2)}, t_I^{(2)}, \\
& \mathbf{y}^{(2)}(t_F^{(2)}), \mathbf{u}^{(2)}(t_F^{(2)}), \mathbf{p}^{(2)}, t_F^{(2)}, \\
& \cdots \\
& \mathbf{y}^{(N)}(t_I^{(N)}), \mathbf{u}^{(N)}(t_I^{(N)}), \mathbf{p}^{(N)}, t_I^{(N)}, \\
& \mathbf{y}^{(N)}(t_F^{(N)}), \mathbf{u}^{(N)}(t_F^{(N)}), \mathbf{p}^{(N)}, t_F^{(N)} \Big] \\
& + \sum_{j=1}^{N} \left\{ \int_{t_I^{(j)}}^{t_F^{(j)}} w^{(j)} \left[\mathbf{y}^{(j)}(t), \mathbf{u}^{(j)}(t), \mathbf{p}^{(j)}, t \right] dt \right\}.
\end{aligned}
\tag{4.40}
$$

Notice that, like the boundary conditions, the objective function may depend on quantities computed in each of the N phases. Furthermore, the objective function includes contributions evaluated at the phase boundaries (point functions) and over the phase (quadrature functions). As written, (4.40) is known as the *problem of Bolza*. When the function $\phi \equiv 0$ in the objective, we refer to this as the *problem of Lagrange* or, if there are no integral terms $w^{(j)} \equiv 0$, the optimization is termed the *problem of Mayer*. It is worth noting that it is relatively straightforward to pose the problem in an alternative form if desirable. For example, suppose the problem is stated in the Lagrange form with a performance index

$$
J = \int_{t_I}^{t_F} w \left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t \right] dt. \tag{4.41}
$$

By introducing an additional state variable, say y_{n+1}, and the differential equation

$$
\dot{y}_{n+1} = w[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t] \tag{4.42}
$$

with initial condition $y_{n+1}(t_I) = 0$, it is possible to replace the original objective function (4.41) with

$$
J = \phi[\mathbf{y}(t_F)] = y_{n+1}(t_F). \tag{4.43}
$$

In fact, for notational simplicity, it is common to define the optimal control problem in the Mayer form. On the other hand, it may not be desirable to make this transformation for computational reasons because adding a state variable y_{n+1} increases the size of the NLP subproblem after discretization. For this reason, the SOCS [25] software implementation treats problems stated in Bolza, Lagrange, or Mayer form. Efficient treatment of quadrature equations will be discussed in Section 4.9.

For clarity, we drop the phase-dependent notation from the remaining discussion; however, it is important to remember that many complex problem descriptions require different dynamics and/or constraints within each phase, and the approach accommodates this requirement.

4.5 Direct Transcription Formulation

The fundamental ingredients required for solving the optimal control problem by direct transcription are now in place. In Section 3.5, we introduced a number of methods for

solving initial value problems. All approaches divide the phase duration (for a single phase) into n_s segments or intervals

$$t_I = t_1 < t_2 < \cdots < t_M = t_F, \tag{4.44}$$

where the points are referred to as node, mesh, or grid points. Define the number of mesh points as $M \equiv n_s + 1$. As before, we use $\mathbf{y}_k \equiv \mathbf{y}(t_k)$ to indicate the value of the state variable at a grid point. However, the methods for solving the IVP described in Section 3.5 did not involve algebraic (control) variables. To extend the methods to control problems, let us denote the control at a grid point by $\mathbf{u}_k \equiv \mathbf{u}(t_k)$. In addition, some discretization schemes require values for the control variable at the midpoint of an interval, and we denote this quantity by $\overline{\mathbf{u}}_k \equiv \mathbf{u}(\bar{t})$ with $\bar{t} = \frac{1}{2}(t_k + t_{k-1})$. To be consistent, we also denote $\mathbf{f}_k \equiv \mathbf{f}[\mathbf{y}(t_k), \mathbf{u}(t_k), \mathbf{p}, t_k]$. It should be emphasized that the subscript k refers to a grid point within a phase.

The basic notion is now quite simple. Let us treat the values of the state and control variables as a set of NLP variables. The differential equations will be replaced by a finite set of defect constraints. As a result of the transcription, the optimal control constraints (4.33)–(4.34) are replaced by the NLP constraints

$$\mathbf{c}_L \leq \mathbf{c}(\mathbf{x}) \leq \mathbf{c}_U, \tag{4.45}$$

where

$$\mathbf{c}(\mathbf{x}) = \left[\boldsymbol{\zeta}_1, \boldsymbol{\zeta}_2, \ldots, \boldsymbol{\zeta}_{M-1}, \boldsymbol{\psi}_I, \boldsymbol{\psi}_F, \mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_M\right]^\top \tag{4.46}$$

with

$$\mathbf{c}_L = \left[\mathbf{0}, \ldots, \mathbf{0}, \mathbf{g}_L, \ldots, \mathbf{g}_L\right]^\top \tag{4.47}$$

and a corresponding definition of $\mathbf{c}_U$. The first $n_y n_s$ equality constraints require that the defect vectors from each of the n_s segments be zero, thereby approximately satisfying the differential equations (4.33). The boundary conditions are enforced directly by the equality constraints on $\boldsymbol{\psi}$ and the nonlinear path constraints are imposed at the grid points. Note that nonlinear equality path constraints are enforced by setting $\mathbf{c}_L = \mathbf{c}_U$. In a similar fashion, the state and control variable bounds (4.35) and (4.36) become simple bounds on the NLP variables. The path constraints and variable bounds are always imposed at the grid points for all discretization schemes. For the Hermite–Simpson and Runge–Kutta discretization methods, the path constraints and variable bounds are also imposed at the interval midpoints.

For the sake of reference, let us summarize the variables and constraints for each of the implicit Runge–Kutta schemes introduced in Section 3.5.

Euler Method

Variables:

$$\mathbf{x}^\top = (\mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M). \tag{4.48}$$

Defects:

$$\boldsymbol{\zeta}_k = \mathbf{y}_{k+1} - \mathbf{y}_k - h_k \mathbf{f}_k. \tag{4.49}$$

Classical Runge–Kutta Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \overline{\mathbf{u}}_2, \ldots, \overline{\mathbf{u}}_M, \mathbf{y}_M, \mathbf{u}_M). \tag{4.50}$$

Defects:

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{1}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right), \tag{4.51}$$

where

$$\mathbf{k}_1 = h_k \mathbf{f}_k, \tag{4.52}$$

$$\mathbf{k}_2 = h_k \mathbf{f}\left(\mathbf{y}_k + \frac{1}{2}\mathbf{k}_1, \overline{\mathbf{u}}_{k+1}. t_k + \frac{h_k}{2}\right), \tag{4.53}$$

$$\mathbf{k}_3 = h_k \mathbf{f}\left(\mathbf{y}_k + \frac{1}{2}\mathbf{k}_2, \overline{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right), \tag{4.54}$$

$$\mathbf{k}_4 = h_k \mathbf{f}(\mathbf{y}_k + \mathbf{k}_3, \mathbf{u}_{k+1}, t_{k+1}). \tag{4.55}$$

Trapezoidal Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M). \tag{4.56}$$

Defects:

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}\left(\mathbf{f}_k + \mathbf{f}_{k+1}\right). \tag{4.57}$$

Hermite–Simpson Method

Variables:

$$\mathbf{x}^{\mathsf{T}} = (\mathbf{y}_1, \mathbf{u}_1, \overline{\mathbf{u}}_2, \ldots, \overline{\mathbf{u}}_M, \mathbf{y}_M, \mathbf{u}_M). \tag{4.58}$$

Defects:

$$\zeta_k = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left(\mathbf{f}_k + 4\overline{\mathbf{f}}_{k+1} + \mathbf{f}_{k+1}\right), \tag{4.59}$$

where

$$\overline{\mathbf{y}}_{k+1} = \frac{1}{2}(\mathbf{y}_k + \mathbf{y}_{k+1}) + \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}), \tag{4.60}$$

$$\overline{\mathbf{f}}_{k+1} = \mathbf{f}\left(\overline{\mathbf{y}}_{k+1}, \overline{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right). \tag{4.61}$$

For simplicity, we have presented all of the schemes assuming that the phase duration is fixed. Of course, in general, the duration of a phase may be variable and it is worthwhile discussing how this changes the discretization. First of all, the set of NLP variables must

be augmented to include the variable time(s) t_I and/or t_F. Second, we must alter the discretization defined by (4.44) such that the stepsize is

$$h_k = \tau_k(t_F - t_I) = \tau_k \Delta t, \tag{4.62}$$

where $\Delta t \equiv (t_F - t_I)$ with constants $0 < \tau_k < 1$ chosen so that the grid points are located at fixed fractions of the total phase duration. Thus, in essence, as the times t_I and/or t_F change, an "accordion"-like grid-point distribution results. Observe that this approach will produce a consistent function generator as described in Section 3.8. Finally, it should also be clear that the set of NLP variables can be augmented to include the parameters p if they are part of the problem description.

4.6 NLP Considerations—Sparsity

4.6.1 Background

In the preceding section, an NLP problem was constructed by discretization of an optimal control problem. In order to solve this NLP problem using any of the methods described in Chapters 1 and 2, it will be necessary to construct the first and second derivatives of the NLP constraints and objective function. With the exception of the boundary conditions, the major portion of the Jacobian matrix is defined as

$$\mathbf{G}_{ij} = \frac{\text{Change in defect constraint on segment } i}{\text{Change in optimization variable at grid point } j}.$$

This matrix will have m rows, where m is the total number of defect constraints, and n columns, where n is the total number of optimization variables. Now as a result of the discretization process, it should be clear that changing a variable at a grid point only affects the nearby constraints. Thus, the derivatives of many of the constraints with respect to the variable are zero. Figure 4.1 illustrates the situation for a simple discretization with 11 grid points. When the variable at grid point 6 is changed, the function connecting points 5 and 6 is altered, as is the function connecting points 6 and 7. However, the remaining portion of the curve is unchanged. In fact, the multiple shooting method described in Section 3.4 was introduced as a way to deal with the "tail wagging the dog" problem found in the simple shooting method (Section 3.3). In essence, reduced sensitivity in the BVP manifests itself mathematically as sparsity in the Jacobian matrix. The whole focus of this section is on how to construct the sparse Jacobian and Hessian matrices efficiently for the transcribed problem.

Example 4.1. In order to motivate the development of the general method, it is convenient to present a simple example problem. Let us consider a problem with four states and one control described by the following state equations:

$$\dot{y}_1 = y_3, \tag{4.63}$$
$$\dot{y}_2 = y_4, \tag{4.64}$$
$$\dot{y}_3 = a\cos u, \tag{4.65}$$
$$\dot{y}_4 = a\sin u. \tag{4.66}$$

The goal is to minimize the time required for a vehicle to move from a fixed initial state to a terminal position by choosing the control $u(t)$. In this particular case, the problem

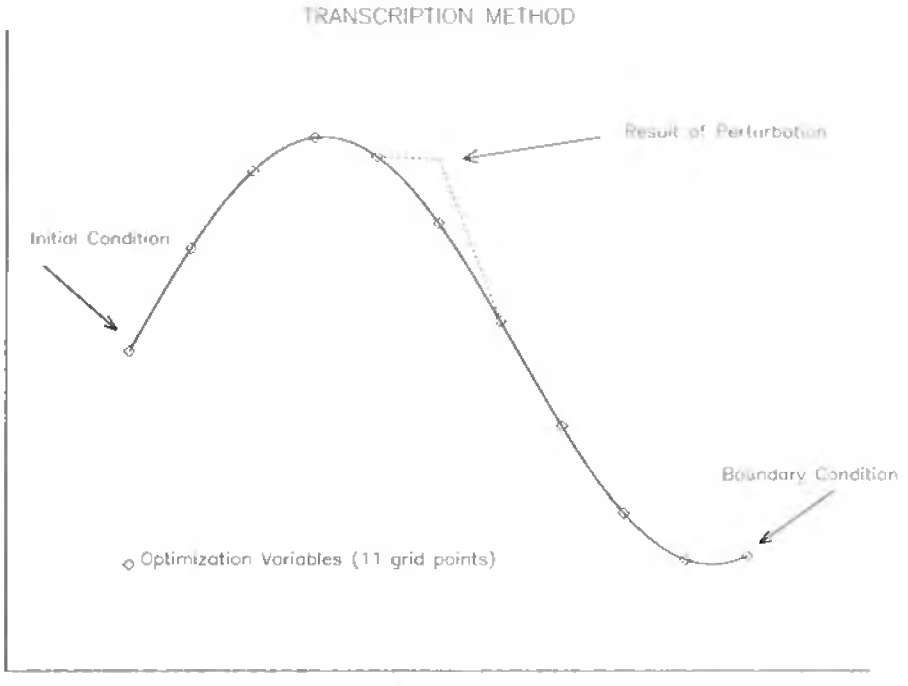


Figure 4.1: Transcription sensitivity.

has an analytic solution referred to as "linear tangent steering" [35]. This also is of considerable practical interest since it is a simplified version of the steering algorithm used by many launch vehicles, including the space shuttle.

4.6.2 Standard Approach

For the sake of comparison, let us begin with the standard direct transcription approach to this problem. The usual technique is to treat the values of the state and control at discrete times as optimization variables, thus leading to the definition

$$\mathbf{x}^{\mathsf{T}} = (t_F, \mathbf{y}_1, \mathbf{u}_1, \ldots, \mathbf{y}_M, \mathbf{u}_M) \tag{4.67}$$

of the NLP variables. The ODEs

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}, \mathbf{u}, t] \tag{4.68}$$

are then approximated by the NLP constraints

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}[\mathbf{f}_{k+1} + \mathbf{f}_k] \equiv \mathbf{c}(\mathbf{x}) \tag{4.69}$$

for $k = 1, \ldots, M-1$. This approximation describes a trapezoidal discretization for M grid points. The Jacobian matrix for the resulting NLP problem is then defined by

$$\mathbf{G} \equiv \frac{\partial \mathbf{c}}{\partial \mathbf{x}}. \tag{4.70}$$

When a finite difference method is used to construct the Jacobian, it is natural to identify the constraint functions as the quantities being differentiated in (2.1). In other words, for the standard approach,

$$\mathbf{q} = \begin{bmatrix} \mathbf{c} \\ F \end{bmatrix} \tag{4.71}$$

and

$$\mathbf{D} = \begin{bmatrix} \mathbf{G} \\ (\nabla F)^{\mathsf{T}} \end{bmatrix}. \tag{4.72}$$

It is also natural to define

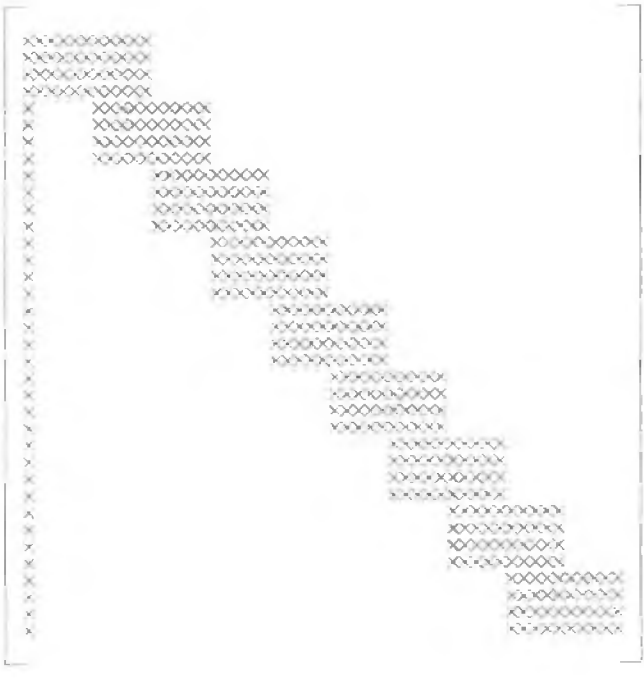
$$\boldsymbol{\omega}^{\mathsf{T}} = (-\lambda_1, \ldots, -\lambda_{M-1}, 1), \tag{4.73}$$

where λ_k are the Lagrange multipliers, so that (2.2)

$$\Omega(\mathbf{x}) = \sum_{i=1}^{m} \omega_i q_i(\mathbf{x}) = F - \sum_{i=1}^{M-1} \lambda_i c_i(\mathbf{x}) = L(\mathbf{x}, \boldsymbol{\lambda}) \tag{4.74}$$

is the *Lagrangian* for the NLP problem. Clearly, it follows that the Hessian of the Lagrangian $\mathbf{H} = \mathbf{E}$, where $\mathbf{E}$ is given by (2.3). For the linear tangent steering example problem, with $M = 10$ grid points, the number of NLP variables is $n = M(n_y + n_u) + 1 = 51$, where $n_y = 4$ is the number of states and $n_u = 1$ is the number of controls. The number of NLP constraints is $m = (M-1)n_y = 36$ and the number of index sets is $\gamma = 2 * (n_y + n_u) + 1 = 11$. Using the notation struct($\mathbf{G}$) to denote the structure of $\mathbf{G}$, the resulting sparse Jacobian is of the form

$$\text{struct}(\mathbf{G}) = \tag{4.75}$$



4.6.3 Discretization Separability

Examination of the expression for the trapezoidal defect (4.69) suggests that it may be desirable to simply group the terms by grid point, i.e.,

$$\begin{aligned} \mathbf{0} &= \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}\left[\mathbf{f}_{k+1} + \mathbf{f}_k\right] \\ &= \left[\mathbf{y}_{k+1} - \frac{h_k}{2}\mathbf{f}_{k+1}\right] + \left[-\mathbf{y}_k - \frac{h_k}{2}\mathbf{f}_k\right]. \end{aligned} \tag{4.76}$$

In so doing it is possible to write the NLP constraints as

$$\mathbf{c}(\mathbf{x}) = \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.77}$$

where

$$\mathbf{B} = \begin{bmatrix} \mathbf{I} & \mathbf{I} & & & \\ & \mathbf{I} & \mathbf{I} & & \\ & & \ddots & & \\ & & & \mathbf{I} & \mathbf{I} \end{bmatrix} \tag{4.78}$$

and

$$\mathbf{q}(\mathbf{x}) = \begin{bmatrix} -\mathbf{y}_1 - \frac{h_1}{2}\mathbf{f}_1 \\ \mathbf{y}_2 - \frac{h_1}{2}\mathbf{f}_2 \\ -\mathbf{y}_2 - \frac{h_2}{2}\mathbf{f}_2 \\ \mathbf{y}_3 - \frac{h_2}{2}\mathbf{f}_3 \\ \vdots \\ -\mathbf{y}_{M-1} - \frac{h_{M-1}}{2}\mathbf{f}_{M-1} \\ \mathbf{y}_M - \frac{h_{M-1}}{2}\mathbf{f}_M \end{bmatrix}. \tag{4.79}$$

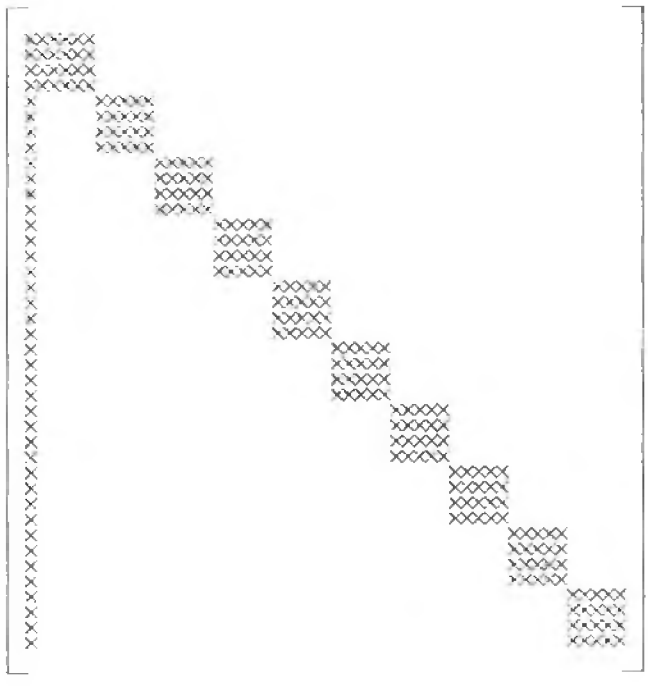
It is then possible to construct sparse difference estimates for the matrix

$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} \tag{4.80}$$

and then construct the NLP Jacobian using

$$\mathbf{G} \equiv \frac{\partial \mathbf{c}}{\partial \mathbf{x}} = \mathbf{B}\mathbf{D}. \tag{4.81}$$

The reason for writing the equations in this manner becomes clear when examining the structure of the matrix $\mathbf{D}$:

$\mathbf{struct(D)} =$  (4.82)

Notice that the matrix $\mathbf{D}$ has approximately half as many nonzero elements per row as the matrix $\mathbf{G}$. Consequently, the number of index sets needed to construct $\mathbf{D}$ is $\gamma = 6$, as compared to $\gamma = 11$ for the standard approach. Thus, by exploiting separability, the Jacobian computation cost has been reduced by nearly a factor of two, and the Hessian cost is reduced by nearly a factor of three.

4.6.4 Right-Hand Side Sparsity (Trapezoidal)

The computation savings observed by exploiting separability suggest further benefit may accrue by grouping terms and also isolating the linear terms, that is,

$$\begin{aligned} \mathbf{0} &= \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{2}[\mathbf{f}_{k+1} + \mathbf{f}_k] \\ &= [\mathbf{y}_{k+1} - \mathbf{y}_k] - \frac{1}{2}\tau_k[t_F\mathbf{f}_{k+1}] - \frac{1}{2}\tau_k[t_F\mathbf{f}_k], \end{aligned} \tag{4.83}$$

where $h_k = \tau_k(t_F - t_I) = \tau_k \Delta t$ with $0 < \tau_k < 1$. Using this construction, the NLP constraints are

$$\mathbf{c}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.84}$$

where the constant matrices $\mathbf{A}$ and $\mathbf{B}$ are given by

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & -\mathbf{I} & \mathbf{0} & \mathbf{I} & & & \\ & & & -\mathbf{I} & \mathbf{0} & \mathbf{I} & \\ \vdots & & & & & & \ddots \end{bmatrix} \tag{4.85}$$

and

$$\mathbf{B} = -\frac{1}{2}\begin{bmatrix} \tau_1\mathbf{I} & \tau_1\mathbf{I} & & & & \\ & & \tau_2\mathbf{I} & \tau_2\mathbf{I} & & \\ & & & & \ddots & \\ & & & & \tau_{M-1}\mathbf{I} & \tau_{M-1}\mathbf{I} \end{bmatrix} \tag{4.86}$$

with the nonlinear relationships isolated in the vector

$$\mathbf{q} = \begin{bmatrix} \Delta t\,\mathbf{f}_1 \\ \Delta t\,\mathbf{f}_2 \\ \vdots \\ \Delta t\,\mathbf{f}_M \end{bmatrix}. \tag{4.87}$$

As before, we can construct sparse finite difference estimates for the matrix

$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} = \left[\begin{array}{c|ccc} \frac{\partial}{\partial t_F}(\Delta t\,\mathbf{f}_1) & \Delta t\frac{\partial \mathbf{f}_1}{\partial \mathbf{x}_1} & & \\ \frac{\partial}{\partial t_F}(\Delta t\,\mathbf{f}_2) & & \Delta t\frac{\partial \mathbf{f}_2}{\partial \mathbf{x}_2} & \\ \vdots & & & \ddots \\ \frac{\partial}{\partial t_F}(\Delta t\,\mathbf{f}_M) & & & \Delta t\frac{\partial \mathbf{f}_M}{\partial \mathbf{x}_M} \end{array}\right] \tag{4.88}$$

and then form the NLP Jacobian

$$\mathbf{G} = \mathbf{A} + \mathbf{B}\mathbf{D}. \tag{4.89}$$

An important aspect of this construction now becomes evident. Notice that the matrix $\mathbf{D}$ in (4.88) involves partial derivatives of the right-hand side functions $\mathbf{f}$ with respect to the state and control, all evaluated at the same grid point. In particular, let us define the nonzero pattern

$$\text{struct}\left(\frac{\partial \mathbf{f}_k}{\partial \mathbf{x}_k}\right) = \text{struct}\left(\frac{\partial \mathbf{f}_k}{\partial \mathbf{y}_k}\middle|\frac{\partial \mathbf{f}_k}{\partial \mathbf{u}_k}\right) \tag{4.90}$$

as the *sparsity template*. The nonzero pattern defined by the sparsity template appears repeatedly in the matrix $\mathbf{D}$ at every grid point introduced by the discretization method. For the linear tangent steering example, the right-hand side sparsity template is of the form

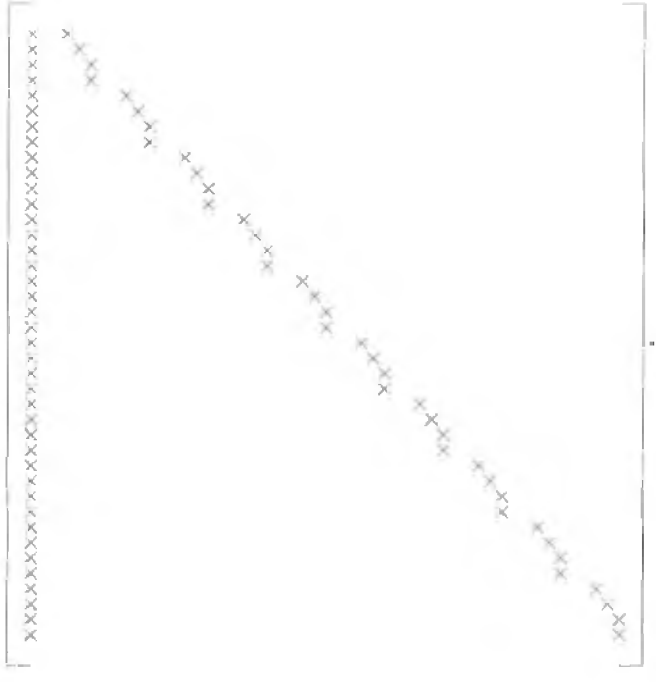
$$\text{struct}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\middle|\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right) = \left[\begin{array}{cccc|c} 0 & 0 & \times & 0 & 0 \\ 0 & 0 & 0 & \times & 0 \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times \end{array}\right]. \tag{4.91}$$

Of course, in general, this right-hand side sparsity template is problem dependent since it is defined by the functional form of the right-hand sides of the state equations (4.68). From a practical point of view, there are a number of alternatives for specifying the right-hand side sparsity. One approach is to simply require the analyst to supply this

information when defining the functions $\mathbf{f}$. A second alternative is to construct the information using some type of automatic differentiation of the user-provided software. The third approach, which is used in the implementation of our software [25], is to numerically construct the right-hand side template using *random* perturbations about *random* nominal points.

Regardless of the approach used to construct the sparsity template, this information can be used to considerable advantage when constructing the matrix $\mathbf{D}$. For our linear tangent steering example,

$$\text{struct}(\mathbf{D}) = \tag{4.92}$$



Comparing (4.82) with (4.92), it is clear that the repeated dense blocks along the diagonal in (4.82) have been replaced with repeated blocks with the right-hand side sparsity template. The net result is that the matrix $\mathbf{D}$ can now be computed with $\gamma = 2$ index sets. To recapitulate, for this linear tangent steering example, one finds the following results:

	Dense	Sparse	Reduction (%)
Number of index sets, γ	11	2	−81.8%
Perturbations per Jacobian/Hessian	77	5	−93.5%

4.6.5 Hermite–Simpson (Compressed) (HSC)

In the previous section, the development focused on exploiting sparsity when the differential equations are approximated using a trapezoidal discretization that is second order, i.e., $\mathcal{O}(h^2)$. To accurately represent the solution, it may be desirable to use a discretization of higher order.

The discretization most widely used in the direct transcription algorithm is Hermite–Simpson, which is of order four, i.e., $\mathcal{O}(h^4)$. For this method, the defect constraints are

given by

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left[\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k\right], \tag{4.93}$$

where

$$\bar{\mathbf{f}}_{k+1} = \mathbf{f}\left[\bar{\mathbf{y}}_{k+1}, \bar{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right], \tag{4.94}$$

$$\bar{\mathbf{y}}_{k+1} = \frac{1}{2}(\mathbf{y}_{k+1} + \mathbf{y}_k) + \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}) \tag{4.95}$$

with $h_k = \tau_k \Delta t$. In order to emphasize the implicit nature of this method, it is instructive to substitute (4.94) and (4.95) into (4.93), yielding

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{h_k}{6}\left\{\mathbf{f}_{k+1} + 4\mathbf{f}\left[\frac{1}{2}(\mathbf{y}_{k+1} + \mathbf{y}_k) + \frac{h_k}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}), \bar{\mathbf{u}}_{k+1}, t_k + \frac{h_k}{2}\right] + \mathbf{f}_k\right\} \tag{4.96}$$

We shall refer to this as the *compressed* form of the Hermite-Simpson method because the Hermite interpolant (4.95) is used to locally eliminate the midpoint state. For this discretization, the NLP variables are

$$\mathbf{x}^\mathsf{T} = (t_F, \mathbf{y}_1, \mathbf{u}_1, \boxed{\bar{\mathbf{u}}_2}, \ldots, \boxed{\bar{\mathbf{u}}_M}, \mathbf{y}_M, \mathbf{u}_M). \tag{4.97}$$

Notice that the values for the control variable at the interval midpoints $\bar{\mathbf{u}}_k$ are introduced as NLP variables, while the corresponding values for the state are not NLP variables.

Proceeding in a manner analogous to the trapezoidal method, the NLP constraints can be written as

$$\mathbf{c}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.98}$$

where

$$\mathbf{q} \equiv \begin{bmatrix} \Delta t\ \left(\mathbf{f}_2 + 4\bar{\mathbf{f}}_2 + \mathbf{f}_1\right) \\ \Delta t\ \left(\mathbf{f}_3 + 4\bar{\mathbf{f}}_3 + \mathbf{f}_2\right) \\ \vdots \\ \Delta t\ \left(\mathbf{f}_M + 4\bar{\mathbf{f}}_M + \mathbf{f}_{M-1}\right) \end{bmatrix}. \tag{4.99}$$

Let us define

$$\mathbf{v}_k = \Delta t\left(\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k\right). \tag{4.100}$$

The derivative matrix is then given by

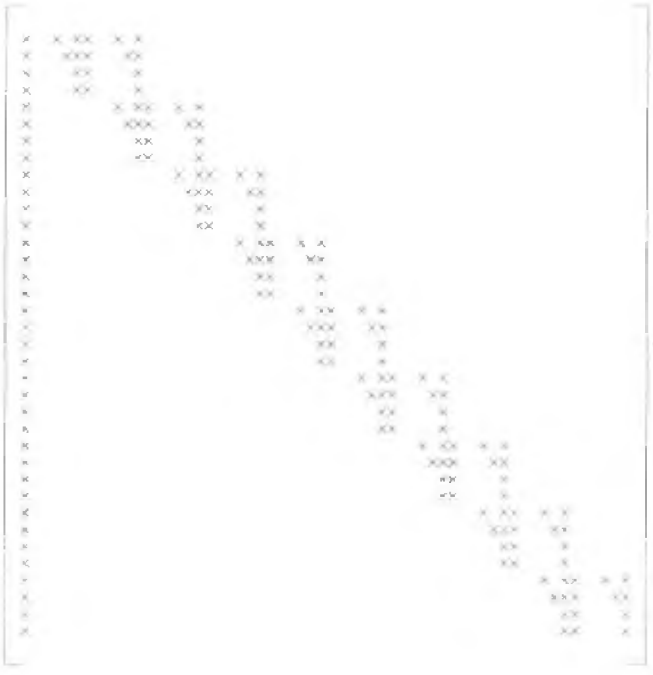
$$\mathbf{D} \equiv \frac{\partial \mathbf{q}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial}{\partial t_F}(\mathbf{v}_1) & \frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{v}_1}{\partial \bar{\mathbf{u}}_2} & \frac{\partial \mathbf{v}_1}{\partial \mathbf{x}_2} & & & \\ \frac{\partial}{\partial t_F}(\mathbf{v}_2) & & & \frac{\partial \mathbf{v}_2}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{v}_2}{\partial \bar{\mathbf{u}}_3} & \frac{\partial \mathbf{v}_2}{\partial \mathbf{x}_3} & \\ \vdots & & & & & & \ddots \\ \frac{\partial}{\partial t_F}(\mathbf{v}_{M-1}) & & & & & & \frac{\partial \mathbf{v}_{M-1}}{\partial \mathbf{x}_M} \end{bmatrix}. \tag{4.101}$$

As in the previous section, it is tempting to construct a template for the repeated sparse blocks in the matrix $\mathbf{D}$. In this case, the sparsity template is

$$\text{struct}\left(\frac{\partial \mathbf{v}_k}{\partial \mathbf{x}_k} \middle| \frac{\partial \mathbf{v}_k}{\partial \bar{\mathbf{u}}_{k+1}} \middle| \frac{\partial \mathbf{v}_k}{\partial \mathbf{x}_{k+1}}\right) = \left[\begin{array}{cccc|c|c|cccc|c} 0 & 0 & \times & 0 & \times & \times & 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & \times & \times & \times & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times & 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times & \times & 0 & 0 & 0 & 0 & \times \end{array}\right]. \tag{4.102}$$

For the linear tangent steering example, we find that

$$\text{struct}(\mathbf{D}) = \tag{4.103}$$



It is now worth comparing these results with the trapezoidal method described in the previous section. First, notice that the template (4.102) has more nonzeros than the trapezoidal template (4.91). This can be attributed to the fact that the function $\mathbf{v}$ in (4.100) is a *nonlinear* combination of the right-hand side functions $\mathbf{f}$. Second, notice that it is not possible to uncouple the neighboring grid-point evaluations as was done when constructing (4.82). This leads to repeated blocks in (4.103) that are more than twice as wide as those in the trapezoidal method. The net result is that the HSC discretization requires $\gamma = 6$, whereas the trapezoidal method needs only $\gamma = 2$.

4.6.6 Hermite–Simpson (Separated) (HSS)

An apparent shortcoming of the compressed form of the Hermite–Simpson discretization is the fact that sparsity in the right-hand side of the differential equations is not fully exploited. Essentially, this is due to the local elimination of the state variable at the midpoint of each interval, i.e., at $t_k + h_k/2$, which implicitly couples neighboring grid points. This difficulty can be avoided by explicitly introducing the local elimination variables and constraints. Thus, instead of (4.93), the discretization constraints *without* local compression are

$$\mathbf{0} = \bar{\mathbf{y}}_{k+1} - \frac{1}{2}(\mathbf{y}_{k+1} + \mathbf{y}_k) - \frac{\tau_k \Delta t}{8}(\mathbf{f}_k - \mathbf{f}_{k+1}) \qquad (\textit{Hermite}), \tag{4.104}$$

$$\mathbf{0} = \mathbf{y}_{k+1} - \mathbf{y}_k - \frac{\tau_k \Delta t}{6}\left[\mathbf{f}_{k+1} + 4\bar{\mathbf{f}}_{k+1} + \mathbf{f}_k\right] \qquad (\textit{Simpson}). \tag{4.105}$$

The first constraint defines the Hermite interpolant for the state at the interval midpoint, while the second constraint enforces the Simpson quadrature over the interval. It is also necessary to introduce additional NLP variables, namely, the state variables at the midpoint of each interval, leading to the augmented set

$$\mathbf{x}^\mathsf{T} = \left(t_F, \mathbf{y}_1, \mathbf{u}_1, \boxed{\bar{\mathbf{y}}_2}, \boxed{\bar{\mathbf{u}}_2}, \mathbf{y}_2, \mathbf{u}_2, \ldots, \boxed{\bar{\mathbf{y}}_M}, \boxed{\bar{\mathbf{u}}_M}, \mathbf{y}_M, \mathbf{u}_M\right). \tag{4.106}$$

As before, the NLP constraints are

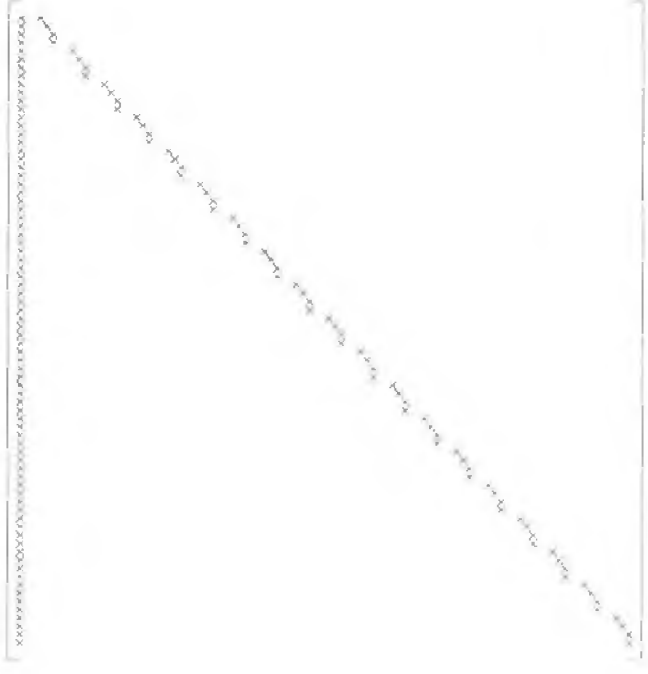
$$\mathbf{c}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.107}$$

where $\mathbf{A}$ and $\mathbf{B}$ are constant matrices and

$$\mathbf{q} = \begin{bmatrix} \Delta t\, \mathbf{f}_1 \\ \Delta t\, \bar{\mathbf{f}}_2 \\ \Delta t\, \mathbf{f}_2 \\ \vdots \\ \Delta t\, \mathbf{f}_{M-1} \\ \Delta t\, \bar{\mathbf{f}}_M \\ \Delta t\, \mathbf{f}_M \end{bmatrix}. \tag{4.108}$$

Thus, by increasing the number of NLP variables and constraints, the sparsity properties of the trapezoidal method have been retained and we find

$$\text{struct}(\mathbf{D}) = \tag{4.109}$$



It is not surprising that the separated Hermite–Simpson formulation requires $\gamma = 2$ index sets just like the trapezoidal, which is clear when comparing (4.109) with (4.92). In effect, this formulation has introduced additional variables and constraints at the interval midpoints.

4.6.7 k-stage Runge–Kutta Schemes

All of the discretizations described are particular examples of k-stage Runge–Kutta schemes introduced in Section 3.5. It is worthwhile to outline how the sparsity-exploiting techniques can be extended. In general, the k-stage Runge–Kutta scheme (3.10)–(3.11) can be written as

$$\mathbf{y}_{k+1} = \mathbf{y}_k + h_k \sum_{j=1}^{k} \beta_j \mathbf{f}_{kj}, \tag{4.110}$$

where

$$\mathbf{y}_{kj} = \mathbf{y}_k + h_k \sum_{\ell=1}^{k} \alpha_{j\ell} \mathbf{f}_{k\ell}, \tag{4.111}$$

$$\mathbf{f}_{kj} = \mathbf{f}\left[\mathbf{y}_{kj}, \mathbf{u}_{kj}, t_{kj}\right], \tag{4.112}$$

$$\mathbf{u}_{kj} = \mathbf{u}\left(t_{kj}\right), \tag{4.113}$$

$$t_{kj} = t_k + h_k \rho_j \tag{4.114}$$

for $1 \leq j \leq k$. The variables at the grid points $(\mathbf{y}_k, \mathbf{u}_k)$ are termed *global* since they must be determined simultaneously over the entire interval $t_I \leq t \leq t_F$. The other variables, namely $(\mathbf{y}_{kj}, \mathbf{u}_{kj})$, are *local* to the interval $t_k \leq t \leq t_{k+1}$. The usual approach is to eliminate the local variables—a process called *parameter condensation*. For the Hermite–Simpson method, which is a three-stage implicit Runge–Kutta scheme, the parameter-condensation process yields the NLP constraints (4.96). Unfortunately, the local elimination process is undesirable when sparsity considerations are introduced. Thus, in general, to exploit sparsity for k-stage Runge–Kutta schemes, one should introduce

1. the local variables $(\mathbf{y}_{kj}, \mathbf{u}_{kj})$ as additional NLP variables and
2. the local elimination conditions (4.111) as additional NLP constraints.

4.6.8 General Approach

In the preceding sections, it has been demonstrated that the discrete approximation to the differential equations can be formulated to exploit sparsity in the problem differential equations. Although the discussion has focused on the constraints derived from the ODEs, the concepts extend in a natural way to all of the problem functions in the NLP problem. Thus, for path-constrained optimal control problems written in semi-explicit form:

$$\begin{aligned} \dot{\mathbf{y}} &= \mathbf{f}\left[\mathbf{y}, \mathbf{u}, t\right], \\ \mathbf{g}_\ell &\leq \mathbf{g}\left[\mathbf{y}, \mathbf{u}, t\right] \leq \mathbf{g}_u, \end{aligned}$$

the key notion is to write the complete set of transcribed NLP functions as

$$\begin{bmatrix} \mathbf{c}(\mathbf{x}) \\ F(\mathbf{x}) \end{bmatrix} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{q}(\mathbf{x}), \tag{4.115}$$

where $\mathbf{A}$ and $\mathbf{B}$ are constant matrices and $\mathbf{q}$ involves the nonlinear functions at grid points. Then it is necessary to construct the *sparsity template* for all of the continuous functions (4.39), that is,

$$\mathcal{T} = \text{struct} \left[\begin{array}{c|c} \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{g}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{w}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{w}}{\partial \mathbf{u}} \end{array} \right]. \tag{4.116}$$

Similar information for the nonlinear boundary functions ψ can also be incorporated. From the sparsity template information, it is possible to construct the sparsity for the matrix $\mathbf{D}$ and compute the finite difference index sets. The first derivative information needed to solve the NLP can then be computed from

$$\left[\begin{array}{c} \mathbf{G} \\ (\nabla F)^{\mathsf{T}} \end{array} \right] = \mathbf{A} + \mathbf{BD}. \tag{4.117}$$

It is also easy to demonstrate that the sparsity pattern for the NLP Hessian is a subset of the sparsity for the matrix $(\mathbf{BD})^{\mathsf{T}}(\mathbf{BD})$, which can also be constructed from the known sparsity of $\mathbf{D}$. It follows from (4.115) that the Lagrangian is

$$L(\mathbf{x}, \boldsymbol{\lambda}) = \left[-\boldsymbol{\lambda}^{\mathsf{T}}, 1\right] \left[\begin{array}{c} \mathbf{c} \\ F \end{array}\right] = \left[-\boldsymbol{\lambda}^{\mathsf{T}}, 1\right] \mathbf{Ax} + \left[-\boldsymbol{\lambda}^{\mathsf{T}}, 1\right] \mathbf{Bq} = \boldsymbol{\sigma}^{\mathsf{T}}\mathbf{x} + \boldsymbol{\omega}^{\mathsf{T}}\mathbf{q}. \tag{4.118}$$

Since there is no second derivative contribution from the linear term $\boldsymbol{\sigma}^{\mathsf{T}}\mathbf{x}$, it is then straightforward to compute the actual Hessian matrix using the sparse differencing formulas (2.6) and (2.7) with $\boldsymbol{\omega}^{\mathsf{T}} = [-\boldsymbol{\lambda}^{\mathsf{T}}, 1]\mathbf{B}$.

At this point it should be emphasized that the potential benefits that accrue from exploiting sparsity in the right-hand sides cannot be achieved with methods that integrate the DAEs over multiple steps. Unfortunately, when a standard initial value method is used to integrate a coupled set of DAEs, in general the repeated blocks will be dense, not sparse. So, for example, a multiple shooting algorithm would require as many perturbations as there are state and control variables. In contrast, for the transcription method it is quite common for $\gamma \ll n_y + n_u$! More information on this subject can be found in [27].

4.6.9 Performance Issues

The preceding sections have described an approach for exploiting sparsity to reduce the cost of constructing finite difference derivatives. However, the approach does introduce an issue that can significantly affect the computational performance of the mesh-refinement process. We temporarily defer the discussion of how mesh refinement is achieved to Section 4.7. In particular, when comparing the HSC form to the HSS form, it is not readily obvious which discretization is better. Both are fourth-order methods, i.e., $\mathcal{O}(h^4)$. In general, the number of index sets for the HSS form will be less than for the HSC form, i.e., $\gamma_s \leq \gamma_c$. However, in order to reduce the cost of computing derivatives, it is necessary to introduce additional NLP variables and constraints, thereby increasing the size of the NLP problem, i.e., $n_s > n_c$. Thus, it is fundamental to assess the performance penalty associated with a larger NLP versus the performance benefit associated with reduced

derivative costs. To this end, let us consider the following model for *total cost per NLP iteration*:

$$\begin{aligned} T &= (\text{Finite differences}) + (\text{Linear algebra}) \\ &= \frac{1}{2} N_r \gamma(\gamma+3) T_r + cn^b \\ &\approx c_1 M \gamma(\gamma+3) T_r + c_2 M^b. \end{aligned}$$

Essentially, the total cost per iteration of an NLP is treated as the sum of two terms. The first term is attributed to the cost of computing sparse finite difference derivative approximations. As such, it depends on the number of index sets γ, the number of times the right-hand side functions are evaluated N_r, and the corresponding right-hand side evaluation time T_r. The second term is attributed to the operations performed by the NLP algorithm and, as such, is related to the size of the problem n. This cost model can be rewritten in terms of the common parameter M, which is the number of grid points. Clearly, this formula depends on problem-specific quantities and the discretization method. Let us denote the cost per NLP iteration for the HSS method by T_s. Similarly, let T_c denote the time for the HSC form. Now it is clear that, as the grid becomes large, $M \to \infty$ and the linear algebra cost will dominate. Thus, for large grids, the HSC method is preferable because the problem size will be smaller. In fact, we can find a *crossover grid size* M^* such that $T_s = T_c$. Then, during the mesh-refinement process,

- use HSS when $M < M^*$,
- use HSC when $M > M^*$.

Numerical tests on a set of 50 mesh-refinement problems suggest that the exponent $b \approx 1.9$. These performance tradeoffs are illustrated in Figure 4.2. To be more precise, the 50 problems were run with a value of $b = 1.9$ and the total solution time was recorded. Then the same set of problems was run with different values for b. The change in the total solution time with respect to the reference value at $b = 1.9$ is plotted as a * in Figure 4.2. The percentage change in the number of function evaluations (relative to the reference value at $b = 1.9$) is plotted with a solid line. Thus, by choosing $b = 1.9$, the best overall algorithm performed was obtained while keeping the number of right-hand side evaluations small. Examples 5.8 and 5.9 describe applications with computationally expensive right-hand side evaluations, where the HSS discretization is clearly preferable.

4.6.10 Performance Highlights

Obviously the problem sparsity will dictate how effective the method is when compared to a standard approach. This section describes how the techniques perform on a particular application that can significantly exploit these properties.

Example 4.2. An example describing the optimal control of a heating process is presented by Heinkenschloss [72]. It can be viewed as a simplified model for the heating of a probe in a kiln. The temperature is described by the following nonlinear parabolic partial differential equation (PDE):

$$q(x,t) = (a_1 + a_2 y)\frac{\partial y}{\partial t} - a_3 \frac{\partial^2 y}{\partial x^2} - a_4 \left(\frac{\partial y}{\partial x}\right)^2 - a_4 y \frac{\partial^2 y}{\partial x^2} \tag{4.119}$$

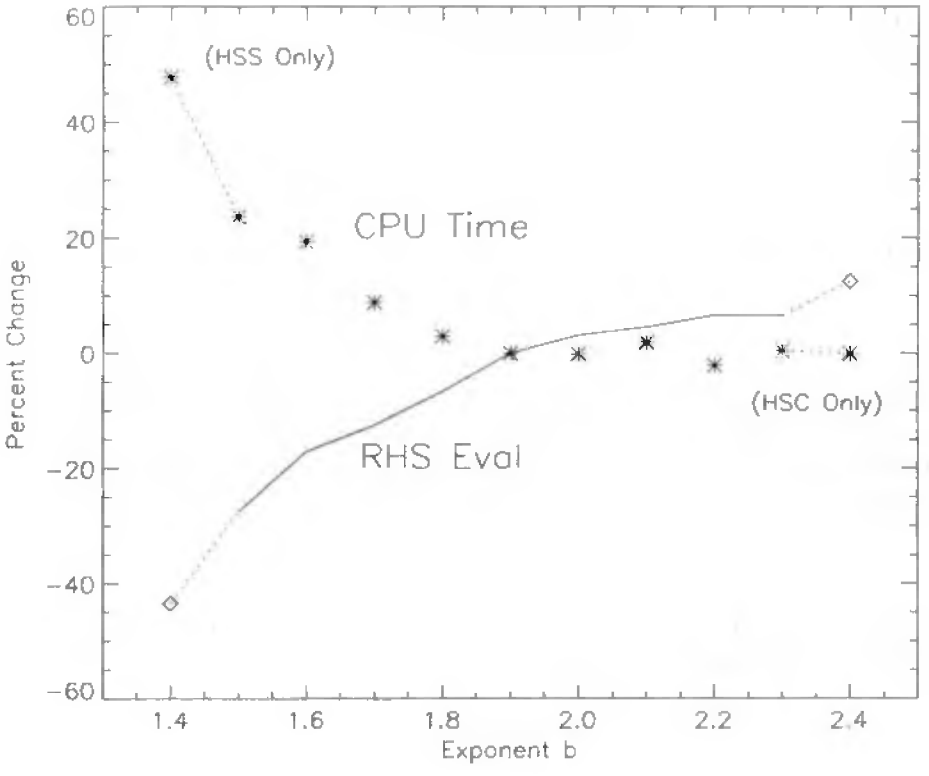


Figure 4.2: Discretization tradeoffs.

with boundary conditions given by

$$(a_3 + a_4 y)\left.\frac{\partial y}{\partial x}\right|_{x=0} = g\left[y(0,t) - u(t)\right], \tag{4.120}$$

$$(a_3 + a_4 y)\left.\frac{\partial y}{\partial x}\right|_{x=1} = 0, \tag{4.121}$$

$$y(x,0) = y_I(x). \tag{4.122}$$

The boundary $x = 1$ is the inside of the probe and $x = 0$ is the outside, where the heat source $u(t)$ is applied. The goal is to minimize the deviation of the temperature from a desired profile, as defined by the objective

$$\phi = \frac{1}{2}\int_0^T \left\{[y(1,t) - y_d(t)]^2 + \gamma u^2(t)\right\} dt, \tag{4.123}$$

by choosing the control function subject to the simple bounds

$$u_L \leq u(t) \leq u_U. \tag{4.124}$$

For consistency with [72], we define the specified functions

$$y_d(t) = 2 - e^{\rho t}, \tag{4.125}$$

$$y_I(x) = 2 + \cos(\pi x), \tag{4.126}$$

$$q(x,t) = \left[\rho(a_1 + 2a_2) + \pi^2(a_3 + 2a_4)\right] e^{\rho t}\cos(\pi x) - a_4\pi^2 e^{2\rho t} + (2a_4\pi^2 + \rho a_2)e^{2\rho t}\cos^2(\pi x) \tag{4.127}$$

and parameter values

$$a_1 = 4, \quad a_2 = 1, \quad a_3 = 4, \quad a_4 = -1, \quad u_U = 0.1,$$
$$\rho = -1, \quad T = 0.5, \quad \gamma = 10^{-3}, \quad g = 1, \quad u_L = -\infty.$$

This distributed parameter optimal control problem can be cast as a lumped parameter control problem using the *method of lines*. The method of lines is an approach for converting a system of PDEs into a system of ODEs. In so doing, the methods of this book become directly applicable. Let us consider a spatial discretization defined by

$$x_i = \frac{i-1}{N-1} \tag{4.128}$$

for $i = 0, 1, \ldots, N, N+1$, where $\delta = 1/(N-1)$. Using this discretization, the partial derivatives are approximated by

$$\frac{\partial y}{\partial x} \approx \frac{1}{2\delta}(y_{i+1} - y_{i-1}), \tag{4.129}$$

$$\frac{\partial^2 y}{\partial x^2} \approx \frac{1}{\delta^2}(y_{i+1} - 2y_i + y_{i-1}). \tag{4.130}$$

After substituting this approximation into the defining relationships and simplifying, one obtains an optimal control problem with the state vector $\mathbf{y}^\mathsf{T} = (y_1, \ldots, y_N)$ and the control vector $\mathbf{v}^\mathsf{T} = (u, y_0, y_{N+1}) = (v_1, v_2, v_3)$. The state equations are

$$\dot{y}_1 = \frac{1}{(a_1 + a_2 y_1)}\left[q_1 + \frac{1}{\delta^2}(a_3 + a_4 y_1)(y_2 - 2y_1 + v_2) + a_4\left(\frac{y_2 - v_2}{2\delta}\right)^2\right], \tag{4.131}$$

$$\dot{y}_i = \frac{1}{(a_1 + a_2 y_i)}\left[q_i + \frac{1}{\delta^2}(a_3 + a_4 y_i)(y_{i+1} - 2y_i + y_{i-1}) + a_4\left(\frac{y_{i+1} - y_{i-1}}{2\delta}\right)^2\right] \tag{4.132}$$

for $i = 2, \ldots, N-1$

$$\dot{y}_N = \frac{1}{(a_1 + a_2 y_N)}\left[q_N + \frac{1}{\delta^2}(a_3 + a_4 y_N)(v_3 - 2y_N + y_{N-1}) + a_4\left(\frac{v_3 - y_{N-1}}{2\delta}\right)^2\right]. \tag{4.133}$$

The boundary conditions (4.120) and (4.121) become path constraints:

$$0 = g(y_1 - v_1) - \frac{1}{2\delta}(a_3 + a_4 y_1)(y_2 - v_2), \tag{4.134}$$

$$0 = \frac{1}{2\delta}(a_3 + a_4 y_N)(v_3 - y_{N-1}) \tag{4.135}$$

and the remaining condition (4.122) defines the initial condition for the states, i.e.,

$$y_i(0) = y_I(x_i). \tag{4.136}$$

Finally, the objective function is just

$$\phi = \frac{1}{2}\int_0^T \left\{ [y_N - y_d]^2 + \gamma v_1^2 \right\} dt. \tag{4.137}$$

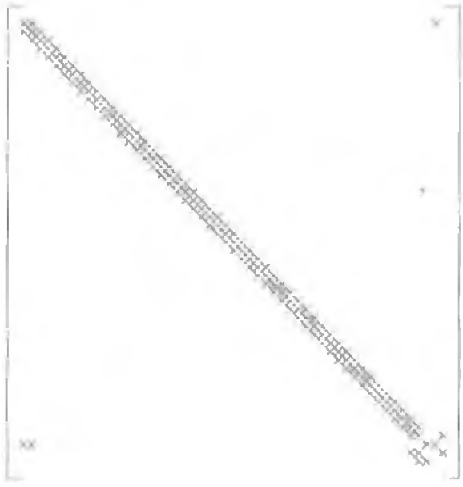
For the results below, $N = 50$, and the state and path equations form a nonlinear index-one DAE system. The optimal solution is illustrated in Figures 4.3 and 4.4. These results were obtained after 5 mesh-refinement iterations (using the algorithm described in Section 4.7), with 164 points in the final time-dependent grid. The final NLP problem involved 9181 variables, 9025 active constraints, and 156 degrees of freedom.

The impact of sparsity on the performance of the algorithm is summarized below:

	Dense	Sparse	Reduction (%)
γ_s	53	4	−92.5%
γ_c	109	15	−86.2%
Func. eval.	42552	983	−97.7%
CPU time	1851.43	377.86	−79.6%

First, notice the dramatic reduction in the number of index sets γ for both HSS and HSC discretization forms. This is due primarily to the right-hand side sparsity template

$$\mathcal{T} = \text{struct}\left[\begin{array}{c|c} \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{g}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{w}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{w}}{\partial \mathbf{u}} \end{array}\right] = \tag{4.138}$$



The comparison is even more dramatic when considering the number of function evaluations needed to solve the problem. If right-hand side sparsity is not exploited, the number of function evaluations is 42552. In comparison, by exploiting sparsity, the problem was solved in 983 evaluations, for a reduction of 97.7%.

4.7 Mesh Refinement

To review, the *transcription method* has three fundamental steps:

1. transcribe the dynamic system into a problem with a *finite* set of variables, then

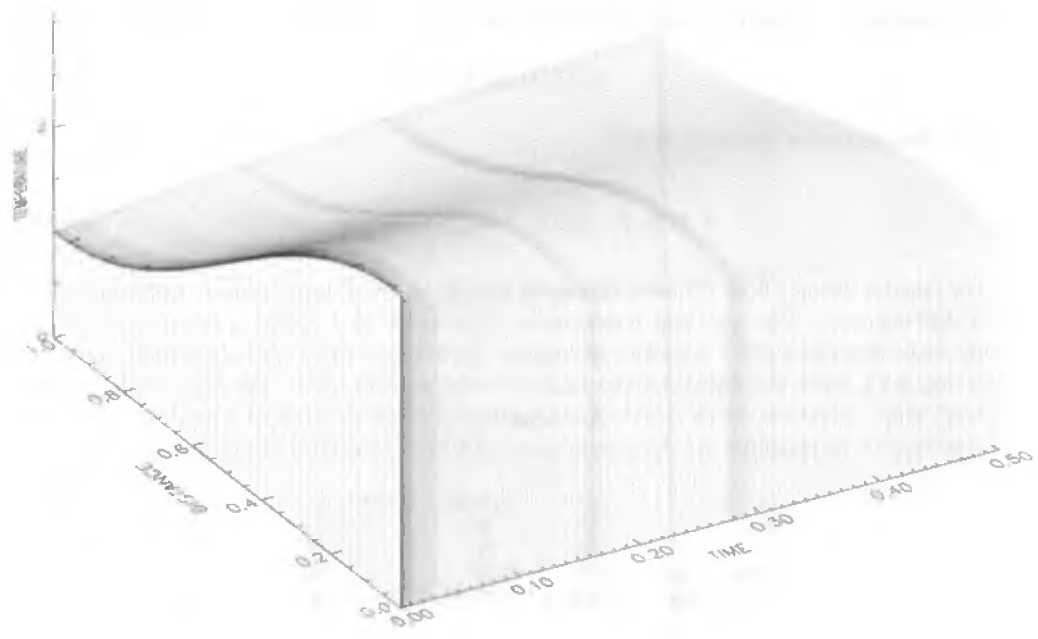


Figure 4.3: Optimal temperature distribution.

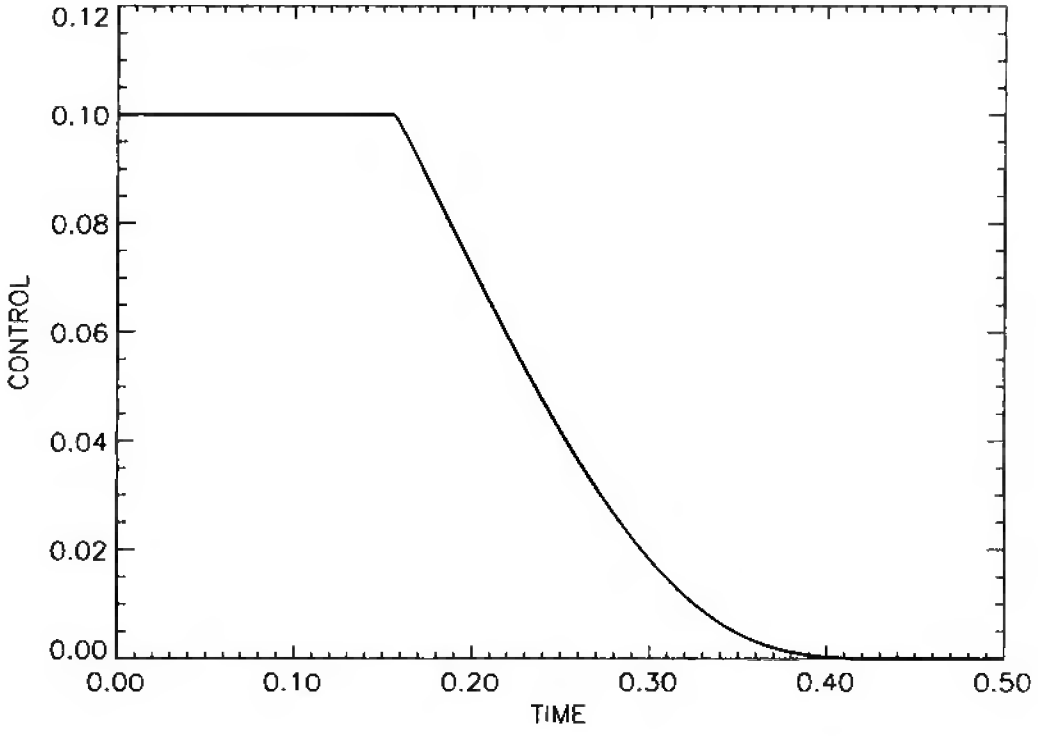


Figure 4.4: Optimal control distribution.

2. solve the finite-dimensional problem using a parameter optimization method (i.e., the NLP subproblem), then

3. assess the accuracy of the finite-dimensional approximation and, if necessary, repeat the transcription and optimization steps.

Techniques for executing step 2 of the transcription method, that is, solving large, sparse NLP problems, are described in Chapter 2. Chapter 3 describes techniques for transcribing the original control problem into a finite-dimensional NLP problem. In the preceding sections, we described how to efficiently compute the sparse Jacobian and Hessian matrices needed by the NLP algorithm. Let us now turn our attention to step 3 in the transcription method, which is called *mesh refinement* or *grid refinement*.

4.7.1 Representing the Solution

The first step in the mesh-refinement process is to construct an approximation to the continuous solution from the information available at the solution of the NLP. Specifically, for the state variable $\mathbf{y}(t)$, let us introduce the approximation

$$\mathbf{y}(t) \approx \tilde{\mathbf{y}}(t) = \sum_{i=1}^{n_1} \gamma_i D_i(t), \tag{4.139}$$

where the functions $D_i(t)$ form a basis for C^1 cubic B-splines with $n_1 = 2M$, where M is the number of mesh points. The coefficients γ_i in the state variable representation are uniquely defined by Hermite interpolation of the discrete solution. Specifically, we require the spline approximation (4.139) to match the state at the grid points

$$\tilde{\mathbf{y}}(t_k) = \mathbf{y}_k \tag{4.140}$$

for $k = 1, \ldots, M$. In addition, we force the derivative of the spline approximation to match the right-hand side of the differential equations, that is, from (4.33),

$$\frac{d}{dt}\tilde{\mathbf{y}}(t_k) = \mathbf{f}_k \tag{4.141}$$

for $k = 1, \ldots, M$.

A similar technique can be used to construct an approximation for the control variables from the discrete data. Specifically, let us introduce the approximation

$$\mathbf{u}(t) \approx \tilde{\mathbf{u}}(t) = \sum_{i=1}^{n_2} \beta_i C_i(t). \tag{4.142}$$

When a trapezoidal discretization is used, the functions $C_i(t)$ form a basis for C^0 piecewise linear B-splines with $n_2 = M$. The coefficients β_i in the control variable representation are uniquely defined by interpolation of the discrete solution. Specifically, we require the spline approximation (4.142) to match the control at the grid points

$$\tilde{\mathbf{u}}(t_k) = \mathbf{u}_k \tag{4.143}$$

for $k = 1, \ldots, M$. When a Hermite–Simpson solution is available, it is possible to use a higher-order approximation for the control. In this case, the functions $C_i(t)$ form a basis

for C^0 quadratic B-splines with $n_2 = 2M - 1$. The coefficients can be defined from the values at the grid points (4.143) as well as the values of the control at the midpoint of the interval, that is,

$$\widetilde{\mathbf{u}}\left[\frac{(t_{k+1}+t_k)}{2}\right] = \overline{\mathbf{u}}_{k+1} \tag{4.144}$$

for $k = 1, \ldots, M-1$.

It is convenient to collect the preceding results in terms of a common B-spline basis. In particular, the continuous functions can be written as

$$\begin{bmatrix} \mathbf{y}(t) \\ \mathbf{u}(t) \end{bmatrix} \approx \begin{bmatrix} \widetilde{\mathbf{y}}(t) \\ \widetilde{\mathbf{u}}(t) \end{bmatrix} = \sum_{i=1}^{N} \boldsymbol{\alpha}_i B_i(t), \tag{4.145}$$

where the functions $B_i(t)$ form a basis for C^0 cubic B-splines with $N = 3M - 2$. It is important to emphasize that, by construction, the state variables are C^1 cubics and the control will be either C^0 quadratic or linear functions, even though they are represented in the space of C^0 cubic B-splines. For a more complete discussion of B-spline approximation, the reader should consult [44]. Is there a reason to prefer a polynomial representation over some other form, such as rational function or Fourier series? Yes! Recall that an implicit Runge–Kutta scheme is a collocation method. The interpolation conditions used to construct the polynomial approximation (in B-spline form) are just the collocation conditions (3.23)–(3.24).

4.7.2 Estimating the Discretization Error

The preceding section describes an approach for representing the functions $\widetilde{\mathbf{y}}(t)$ and $\widetilde{\mathbf{u}}(t)$. The fundamental question is how well do these functions approximate the true solution $\mathbf{y}(t)$ and $\mathbf{u}(t)$. To motivate the discussion, let us reconsider the simplified form of the problem introduced in Section 4.1. Suppose we must choose the control functions $\mathbf{u}(t)$ to minimize (4.1) subject to the state equations (4.2) and the boundary conditions (4.3). The initial conditions $\mathbf{y}(t_I) = \mathbf{y}_I$ are given at the fixed initial time t_I, and the final time t_F is free. As stated, solving the necessary conditions is a two-point BVP in the variables $\mathbf{y}(t)$, $\mathbf{u}(t)$, *and* $\boldsymbol{\lambda}(t)$. In fact, many of the refinement ideas we will discuss are motivated by boundary value methods (cf. [2]).

A direct transcription method does not explicitly form the necessary conditions (4.7)–(4.11). In fact, one of the major reasons direct transcription methods are popular is that it is not necessary to derive expressions for $\mathbf{H}_\mathbf{y}$, $\mathbf{H}_\mathbf{u}$, and $\boldsymbol{\Phi}_\mathbf{y}$ and it is not necessary to estimate values for the adjoint variables $\boldsymbol{\lambda}(t)$. On the other hand, because the adjoint equations are not available, they cannot be used to assess the accuracy of the solution. Consequently, we choose to address a different measure of discretization error. Specifically, we *assume* $\widetilde{\mathbf{u}}(t)$ is correct (and optimal) and estimate the error between $\widetilde{\mathbf{y}}(t)$ and $\mathbf{y}(t)$. This is a subtle, but very important, distinction, for it implies that optimality of the control history $\widetilde{\mathbf{u}}(t)$ is not checked when measuring the discretization error. The functions $\widetilde{\mathbf{u}}(t)$ are constructed to interpolate the discrete values $\mathbf{u}_k$ as described in the previous section. The discrete values $\mathbf{u}_k$ are the solution of an NLP problem and satisfy the NLP (KKT) necessary conditions. However, only in the limit as $h_k \to 0$ do the KKT conditions become equivalent to the necessary conditions (4.7)–(4.11). Computational experience will be presented that tends to corroborate the validity of this approach.

Specifically, let us estimate the error in the state variables as a result of the discretization (4.57) or (4.59). We restrict this analysis to the class of controls that can be represented as C^0 quadratic B-splines. This restriction in turn implies that one can expect to accurately solve an optimal control problem provided:

1. the optimal state variable $\mathbf{y}(t)$ is C^1 and
2. the optimal control variable $\mathbf{u}(t)$ is C^0

within the phase, i.e., for $t_I \leq t \leq t_F$. On the other hand, the solution to the optimal control problem as posed in (4.33)–(4.36) may in fact require discontinuities in the control and/or state derivatives. In particular, when the path constraints do not involve the control variable explicitly, the optimal solution may contain *corners*. Similarly, when the control appears linearly in the differential equations, *bang-bang* control solutions can be expected. Consequently, if the transcription method described is applied to problems of this type, some inaccuracy must be expected unless the locations of discontinuities are introduced explicitly as phase boundaries. Thus, we will be satisfied with accurately solving problems when the control is continuous and the state is differentiable. If this is not true, we will be satisfied if the method "does something reasonable."

When analyzing the behavior of an integration method, it is common to ascribe an *order of accuracy* to the algorithm (cf. [41]). Typically, the solution of an ODE is represented by an expansion of the form

$$\mathbf{y}(t,h) = \mathbf{y}(t) + \sum_{i=p}^{\infty} \mathbf{c}_i(t) h^i. \tag{4.146}$$

The *global error* at a point t_{k+1} is the difference between the computed solution $\mathbf{y}_{k+1}$ and the exact solution $\mathbf{y}(t_{k+1})$. The *local error* is the difference between the computed solution $\mathbf{y}_{k+1}$ and the solution of the differential equation that passes through the computed point $\mathbf{y}_k$. For ODEs, typically if the global error is $\mathcal{O}(h^p)$, then the local error is $\mathcal{O}(h^{p+1})$. Thus, if the order of accuracy of a method is p, the local error for the method at step k is of the form

$$\epsilon_k \approx \|\mathbf{c}_k h^{p+1}\|, \tag{4.147}$$

where the coefficients $\mathbf{c}_k$ typically depend on partial derivatives of the right-hand side $\mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t]$. The Hermite-Simpson discretization (4.59) is of order $p = 4$, while the trapezoidal discretization (4.57) is of order $p = 2$.

The standard order analysis of a system of ODEs is modified when considering a system of DAEs [30]. In particular, when one or more of the path constraints (4.34) is active, the constrained arc is characterized by a DAE of the form

$$\dot{\mathbf{y}} = \mathbf{f}[\mathbf{y}(t), \mathbf{u}(t), t], \tag{4.148}$$

$$0 = \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{4.149}$$

The index of a DAE is one measure of how singular the DAE is. It is known that numerical methods may experience an order reduction when applied to a DAE. When a path constraint becomes active, the index of the DAE may change, and there can be a corresponding change in the order of the method. As a result, for a path-constrained problem, we must assume that (4.147) is replaced by

$$\epsilon_k \approx \|\mathbf{c}_k h^{p-r+1}\|, \tag{4.150}$$

where r is the order reduction. Thus, we expect that for some problems the index and hence the order reduction will change as a function of time t. Unfortunately, in general, the index of the DAE is unknown and, therefore, the order reduction is also not known.

There is a further problem in that the theory for IRK methods applied to DAEs usually leads to different amounts of order reduction in different variables [30]. This is true for both the trapezoidal and Hermite–Simpson methods [37, 77]. In addition, the difference between the local and global errors can sometimes be greater than one. In a complex optimization problem, the activation and deactivation of constraints cannot only change the index but can change what the index of a particular variable is. We distinguish only between the control and the state so that order reduction is always taken to be the largest order reduction in all the state variables. In contrast to most traditional methods, let us estimate the order reduction and use this information to improve the refinement process. Consequently, it is not important to determine why the order is reduced but rather by how much [17, 16].

To estimate the order reduction, we need to first estimate the discretization error on a given mesh. There are a number of ways to do so. As previously stated, in estimating the local error we assume that the computed control is correct. Consider a single interval $t_k \leq t \leq t_k + h_k$, that is, a single integration step. Suppose the NLP problem has produced a spline solution $\widetilde{\mathbf{y}}(t)$ and $\widetilde{\mathbf{u}}(t)$ to the ODEs (4.148). From (4.148),

$$\mathbf{y}(t_k + h_k) = \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \dot{\mathbf{y}} dt = \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \mathbf{f}\left[\mathbf{y}, \mathbf{u}, t\right] dt. \tag{4.151}$$

Observe that this expression for $\mathbf{y}(t_k + h_k)$ involves the true value for both $\mathbf{y}$ and $\mathbf{u}$, which are unknown. Consequently, we may consider the approximation

$$\widehat{\mathbf{y}}(t_k + h_k) \equiv \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \mathbf{f}\left[\widetilde{\mathbf{y}}(t), \widetilde{\mathbf{u}}(t), t\right] dt, \tag{4.152}$$

where the spline solution $\widetilde{\mathbf{y}}(t)$ and $\widetilde{\mathbf{u}}(t)$ appears in the integrand. A second alternative is the expression

$$\widehat{\mathbf{y}}(t_k + h_k) \equiv \mathbf{y}(t_k) + \int_{t_k}^{t_k+h_k} \mathbf{f}\left[\mathbf{y}(t), \widetilde{\mathbf{u}}(t), t\right] dt, \tag{4.153}$$

where the real solution $\mathbf{y}(t)$ and the spline approximation $\widetilde{\mathbf{u}}(t)$ appear in the integrand.

With either (4.152) or (4.153), we can define the *discretization error* on the kth mesh iteration as

$$\eta_k = \max_i \left\{a_i \left|\widetilde{y}_i(t_k + h_k) - \widehat{y}_i(t_k + h_k)\right|\right\} \tag{4.154}$$

for $i = 1, \ldots, n$, where the weights a_i are chosen to appropriately normalize the error.

However, our particular need for these estimates imposes certain special restrictions. First, we want them to be part of a method that will be used on a wide variety of problems, including some problems that are stiff. Second, we want to use the estimates on coarse grids. Unfortunately, an estimate computed from (4.153) based on an explicit integrator may be unstable on coarse grids. While (4.153) might be the most accurate, its computation would require an explicit integration of $\dot{\mathbf{y}} = \mathbf{f}\left[\mathbf{y}, \widetilde{\mathbf{u}}, t\right]$ on a possibly large grid with tight error control. This is particularly unfortunate since both of the primary discretization methods (trapezoidal and Hermite–Simpson) are implicit schemes with

very good stability properties. These special requirements lead us to abandon (4.153) and focus on (4.152). First, let us rewrite (4.152) as

$$\begin{aligned}\widehat{\mathbf{y}}_{k+1} &= \mathbf{y}_k + \int_{t_k}^{t_k+h_k} \widetilde{\mathbf{f}} dt \\ &= \mathbf{y}_k + \int_{t_k}^{t_k+h_k} \dot{\widetilde{\mathbf{y}}} dt - \int_{t_k}^{t_k+h_k} \dot{\widetilde{\mathbf{y}}} dt + \int_{t_k}^{t_k+h_k} \widetilde{\mathbf{f}} dt \\ &= \mathbf{y}_k + \widetilde{\mathbf{y}}_{k+1} - \widetilde{\mathbf{y}}_k - \int_{t_k}^{t_k+h_k} [\dot{\widetilde{\mathbf{y}}} - \widetilde{\mathbf{f}}] dt.\end{aligned}$$

By definition, there is no error at the beginning of the interval for a local error estimate, so we can set $\mathbf{y}_k = \widetilde{\mathbf{y}}_k$, which leads to

$$\widetilde{\mathbf{y}}_{k+1} - \widehat{\mathbf{y}}_{k+1} = \int_{t_k}^{t_k+h_k} [\dot{\widetilde{\mathbf{y}}} - \widetilde{\mathbf{f}}] dt.$$

Taking absolute values of each component, we then obtain the bound

$$|\widetilde{\mathbf{y}}_{i,k+1} - \widehat{\mathbf{y}}_{i,k+1}| = \left| \int_{t_k}^{t_k+h_k} \left[\dot{\widetilde{\mathbf{y}}}_i - \widetilde{\mathbf{f}}_i \right] dt \right| \leq \int_{t_k}^{t_k+h_k} |\dot{\widetilde{\mathbf{y}}}_i - \widetilde{\mathbf{f}}_i| dt.$$

Therefore, let us define the *absolute local error* on a particular step by

$$\eta_{i,k} = \int_{t_k}^{t_{k+1}} |\varepsilon_i(s)| ds, \tag{4.155}$$

where

$$\varepsilon(t) = \dot{\widetilde{\mathbf{y}}}(t) - \mathbf{f}[\widetilde{\mathbf{y}}(t), \widetilde{\mathbf{u}}(t), t] \tag{4.156}$$

defines the error in the differential equation as a function of t. Notice that the arguments of the integrand use the spline approximations (4.145) for the state and control evaluated at intermediate points in the interval. From this expression for the absolute error, we can define the *relative local error* by

$$\epsilon_k \approx \max_i \frac{\eta_{i,k}}{(w_i + 1)}, \tag{4.157}$$

where the scale weight

$$w_i = \max_{k=1}^{M} \left[|\widetilde{y}_{i,k}|, |\dot{\widetilde{y}}_{i,k}| \right] \tag{4.158}$$

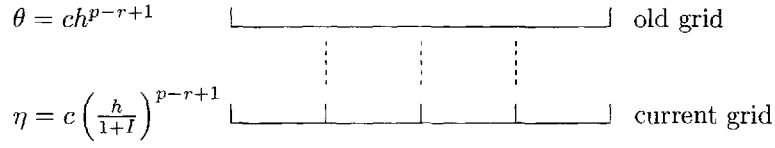
defines the maximum value for the ith state variable or its derivative over the M grid points in the phase. Notice that ϵ_k is the maximum relative error over all components i in the state equations $\dot{\mathbf{y}} - \mathbf{f}$ evaluated in the interval k.

The approximations (4.154) and (4.157) are similar; however, they differ in two respects. The weightings are quite different. Also, (4.154) emphasizes the error in prediction, which is typical with ODE integrators, while (4.157) emphasizes the error in solving the equations. Since the latter is closer to the SOCS termination criteria, we will use (4.157) in the ensuing algorithms.

Now let us consider how to compute an estimate for the error η_k. Since this discretization error is essential for estimating the order reduction, we choose to construct an accurate estimate for the integral (4.155). Because the spline approximations for the state and control are used, the integral (4.155) can be evaluated using a standard quadrature method. In particular, we compute the integral using a Romberg quadrature algorithm with a tolerance close to machine precision.

4.7.3 Estimating the Order Reduction

In order to use the formula (4.150), it is necessary to know the order reduction r. We propose to compute this quantity by comparing the behavior on two successive mesh-refinement iterations. Specifically, assume that the current grid was obtained by subdividing the old grid, that is, the current grid has more points than the old grid. Let us focus on the behavior of a *single* variable on a *single* interval in the old grid as illustrated below:



Denote the discretization error on the old grid by θ. The error on an interval in the old grid is then

$$\theta = ch^{p-r+1}, \tag{4.159}$$

where p is the order of the discretization on the old grid and h is the stepsize for the interval. If the interval on the old grid is subdivided by adding I points, the resulting discretization error is

$$\eta = c\left(\frac{h}{1+I}\right)^{p-r+1}. \tag{4.160}$$

If we assume the order reduction r and the constant c are the same on the old and current grids, then we can solve (4.159) and (4.160) for these quantities.

Solving gives

$$\hat{r} = p + 1 - \frac{\log(\theta/\eta)}{\log(1+I)}. \tag{4.161}$$

Choosing an integer in the correct range, the estimated order reduction is given by

$$r = \max\left[0, \min\left(\text{nint}(\hat{r}), p\right)\right], \tag{4.162}$$

where nint denotes the nearest integer. As a final practical matter, we assume that the order reduction is the same for all $I+1$ subdivisions of the old interval. Thus, if an interval on the old grid was subdivided into three equal parts, we assume the order reduction is the same over all three parts. Thus, the resolution of our order-reduction estimates is dictated by the old, coarse grid.

To summarize, we compare the discretization errors for each interval on the old grid, i.e., θ_k, with the corresponding discretization errors on the current grid. Because

the current grid is constructed by subdividing the old grid. we can then compute the estimated order reduction for all intervals on the current grid.

While (4.161) provides a formula for the observed order reduction, this estimate is sensitive to the computed discretization error estimates η and θ. To appreciate the sensitivity, let us assume q, p are the orders of the Hermite–Simpson discretization on the old and current grids. Generalizing the expression (4.161), define the function

$$Q(a,b) = q + 1 - \frac{\log(a/b)}{\log(1+I)}.$$

Notice that

$$Q(p_1 a, p_2 b) = Q(a,b) - \frac{\log(p_1/p_2)}{\log(1+I)}.$$

Here we are thinking of p_1/p_2 as the ratio in the computed discretization errors. Notice that $p_1/p_2 = 1.07$ with $I = 1$ gives a reduction of 0.1, while $p_1/p_2 = 1.15$ gives a reduction of 0.2. Thus, a change of 15% in the discretization error estimate could easily alter the estimated value of r if it reduced, say, from 1.65 to 1.45. In order to deal with this sensitivity in the mesh refinement, we have done two things:

1. computed the discretization errors using a very accurate quadrature method to evaluate (4.155), and

2. computed the weights (4.158) only once at the end of the first refinement iteration.

4.7.4 Constructing a New Mesh

The purpose of this section is to delineate an approach for constructing a new mesh using information about the discretization error on a current mesh. We will use the terminology "old grid" to refer to the previous refinement iteration, "current grid" to refer to the current iteration. and "new grid" when describing the *next* iteration. Certainly the primary goal is to create a new mesh with less discretization error than the current one. On the other hand, simply adding a large number of points to the current grid increases the size of the NLP problem to be solved. thereby causing a significant computational penalty. Briefly then, the goal is to reduce the discretization error as much as possible using a *specified* number of new points.

To motivate the discussion, let us consider the simple example illustrated in Figure 4.5. which shows the discretization error and the stepsize as a function of the normalized interval τ. Suppose that the old grid has five intervals (six grid points) and the largest discretization error is in interval three. If interval three is subdivided by adding a grid point, the corresponding discretization error should be reduced as illustrated by the dark shaded region in Figure 4.5. Obviously, the process can be repeated. each time adding a point to the interval with the largest discretization error. Thus, a new mesh can be constructed by successively adding points to the intervals with the largest discretization errors.

In the preceding section, an approach was described for computing an error estimate for each segment or interval in the current grid. By equating (4.150) with (4.157), we obtain

$$\|\mathbf{c}_k\| h^{p-r_k+1} = \max_i \frac{\boldsymbol{\eta}_{i,k}}{(w_i+1)} \tag{4.163}$$

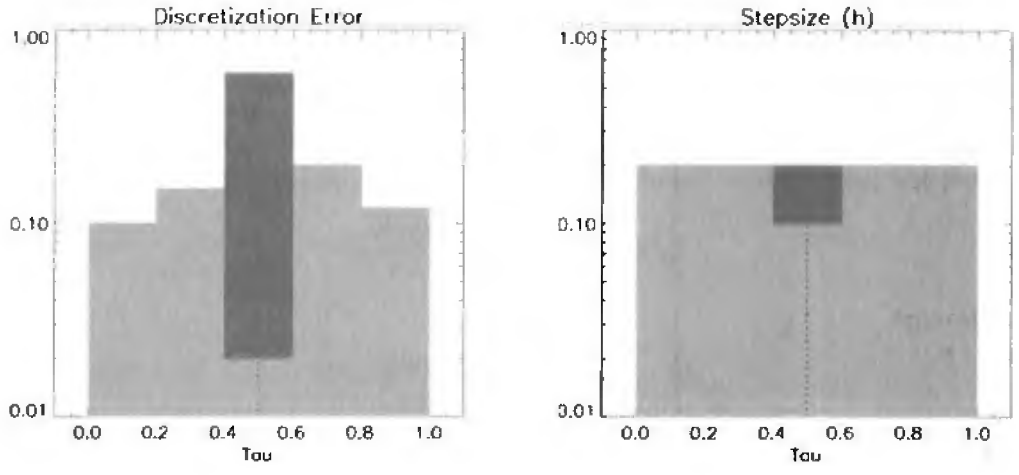


Figure 4.5: Subdividing the grid.

so that

$$\|c_k\| = \max_i \frac{\eta_{i,k}}{(w_i+1)h^{p-r_k+1}}. \tag{4.164}$$

Let us suppose we are going to *subdivide* the current grid, i.e., the new grid will contain the current grid points. Let us define the integer I_k as the number of points to add to interval k, so that from (4.150) and (4.164), we may write

$$\epsilon_k \approx \|c_k\| \left(\frac{h}{1+I_k}\right)^{p-r_k+1} = \max_i \frac{\eta_{i,k}}{(w_i+1)} \left(\frac{1}{1+I_k}\right)^{p-r_k+1} \tag{4.165}$$

for integers $I_k \geq 0$. Specifically, if we add I_k points to interval k in the current mesh, this is an approximation for the error on *each* of the $1+I_k$ subintervals. Then the new mesh can be constructed by choosing the set of integers I_k to minimize

$$\phi(I_k) = \max_k \epsilon_k \tag{4.166}$$

and satisfy the constraints

$$\sum_{k=1}^{n_s} I_k \leq M-1 \tag{4.167}$$

and

$$I_k \leq M_1 \tag{4.168}$$

for $k = 1, \ldots, n_s$. Essentially, we want to minimize the maximum error over all of the intervals in the current mesh by adding at most $M-1$ total points. In addition, the number of points that are added to a single interval is limited to M_1. This restriction is incorporated in order to avoid an excessive number of subdivisions in a single interval. In fact, in our computational implementation we terminate the mesh-refinement process

when a single interval has M_1 points. Equations (4.166)–(4.168) define a nonlinear *integer programming problem.*

This approach formalizes a number of reasonable properties for a mesh-refinement procedure. When the errors on each interval of the current mesh are approximately the same, i.e.,

$$\epsilon_1 \approx \epsilon_2 \approx \cdots \approx \bar{\epsilon}, \tag{4.169}$$

we say that the error is *equidistributed*, where the average error

$$\bar{\epsilon} = \frac{1}{M} \sum_{k=1}^{M} \epsilon_k. \tag{4.170}$$

In this case, the new mesh defined by the integer programming problem (4.166)–(4.168) will simply subdivide each interval in the current mesh. On the other hand, the error for the current mesh may be dominated by the error on a single interval α, i.e.,

$$\epsilon_\alpha \gg \epsilon_k \tag{4.171}$$

for $k = 1, \ldots, n_s$ with $k \neq \alpha$. In this case, the solution to (4.166)–(4.168) will require adding as many as M_1 points into interval α. Typically, we use $M_1 = 5$.

4.7.5 The Mesh-Refinement Algorithm

Let us now summarize the procedure for refinement of the mesh. Denote the mesh-refinement iteration number by j_r. Assume the current grid has M points. The goal of the mesh-refinement procedure is to select the number and location of the grid points in the new mesh as well as the order of the new discretization. Typically, we begin with a low-order discretization and switch to a high-order method at some point in the process. In our software implementation, the default low/high-order pair are trapezoidal and Hermite–Simpson, respectively. The desired error tolerance is δ and we would like the new mesh to be constructed such that it has an error below this tolerance. In fact, when making predictions, we would like the *predicted* errors to be "safely" below, say, $\hat{\delta} = \kappa\delta$, where $0 < \kappa < 1$. Typically, we set $\kappa = 1/10$. The procedure begins with values for the discretization error on all intervals in the current mesh, i.e., ϵ_k for $k = 1, \ldots, n_s$, and we initialize $I_k = 0$.

Mesh-Refinement Algorithm

1. **Construct Continuous Representation.** Compute the cubic spline representation (4.145) from the discrete solution $\mathbf{x}^*$.

2. **Estimate Discretization Error.** Compute an estimate for the discretization error ϵ_k in each segment of the current mesh, that is, evaluate (4.155) using Romberg quadrature: compute the average error from (4.170).

3. **Select Primary Order for New Mesh.**

   (a) If the error is equidistributed for the low-order method, increase the order, i.e., if $p < 4$ and $\epsilon_\alpha \leq 2\bar{\epsilon}$, then set $p = 4$ and terminate.

   (b) Otherwise, if $(p < 4)$ and $j_r > 2$, then set $p = 4$ and terminate.

4. **Estimate Order Reduction.** Compare the current and old grids to compute r_k from (4.161) and (4.162).

5. **Construct New Mesh.**

   (a) Compute the interval α with maximum error, i.e.,

   $$\epsilon_\alpha = \max_k \epsilon_k. \tag{4.172}$$

   (b) Terminate if

   - M' points have been added ($M' \geq \min[M_1, \kappa M]$)

   and

   - the *error is within tolerance:* $\epsilon_\alpha \leq \delta$ and $I_\alpha = 0$ or
   - the *predicted error is safely within tolerance:* $\epsilon_\alpha \leq \kappa\delta$ and $0 < I_\alpha < M_1$ or
   - $M - 1$ points have been added or
   - M_1 points have been added to a *single* interval.

   (c) Add a point to interval α, i.e., $I_\alpha \leftarrow I_\alpha + 1$.

   (d) Update the predicted error for interval α from (4.165).

   (e) Return to step 5a.

Observe that early in the mesh-refinement process, when the discretization error estimates are crude, we limit the growth so that the new mesh will have at most $2M - 1$ points. The intent is to force a new NLP solution, which presumably will lead to better estimates of the error. Furthermore, in step 5b of the procedure, when $I_\alpha \neq 0$, the error ϵ_α is predicted (and presumably less reliable). In this case, we force it to be safely less than the tolerance before stopping. In addition, the refinement is not terminated without adding a minimum number of grid points. This is done to preclude a sequence of refinement iterations that only add one or two points.

The procedure used to modify the order and size of the mesh is somewhat heuristic and is designed to be efficient on most applications. Because the trapezoidal method is both robust and efficient when computing sparse finite difference derivatives, the intent is to solve the initial sparse NLP problem using a trapezoidal discretization. In fact, computational experience demonstrates the value of initiating the process with a coarse mesh and low-order method. On a set of 49 optimal control test problems, on average, the strategy requires 17.1% fewer evaluations of the right-hand side functions $\mathbf{f}$ and $\mathbf{g}$ than a strategy that begins with the Hermite–Simpson discretization. On the other hand, in the software implementation (SOCS) [25], it is possible to specify the initial discretization, which may be effective when the user can provide a good initial guess for the solution. If the discretization error appears to be equidistributed, it is reasonable to switch to a higher-order method (i.e., Hermite–Simpson). However, when the error is badly distributed, at least two different discrete solutions are obtained before the order is increased. The default low-order (trapezoidal) and high-order (Hermite–Simpson) schemes are both IRK methods. The method used for constructing a new mesh always results in a subdivision of the current mesh, which has been found desirable in practice. The minimax approach to adding points is designed to emphasize equidistributing the error when only a limited number of points are included in the new grid.

4.7.6 Computational Experience

The mesh-refinement procedure was tested on a standard set of SOCS test problems. The collection consists of 53 optimal control problems and/or BVPs with path constraints, various degrees of nonlinearity, and computational complexity. For comparison, the same problems were solved using the old mesh-refinement strategy described in [26], which did not estimate the order reduction. The results are summarized below.

Performance Summary (Old versus New)	
Total time decrease (16601.63 versus 12956.20)	−22%
Average time change (all problems)	−0.23%
Maximum time increase (all problems)	109.31%
Minimum time decrease (all problems)	−62.20%

Examination of the results suggests that for most problems, there was little difference in the two techniques. However, since the total time for the test set was noticeably reduced, this also suggests that there was very significant improvement on at least a few of the problems. Clearly, for problems that do not exhibit any significant index reduction, little change can be expected. On the other hand, for *some* "hard" problems, there is apparently a noticeable improvement.

Example 4.3. To illustrate the method, let us consider a problem specifically constructed to be hard. The system is defined by the following DAEs:

$$\begin{aligned}
\dot{y}_1 &= -10y_1 + u_1 + u_2, \\
\dot{y}_2 &= -2y_2 + u_1 + 2u_2, \\
\dot{y}_3 &= -3y_3 + 5y_4 + u_1 - u_2, \\
\dot{y}_4 &= 5y_3 - 3y_4 + u_1 + 3u_2, \\
y_1^2 + y_2^2 + y_3^2 + y_4^2 &\geq 3p(t,3,12) + 3p(t,6,10) + 3p(t,10,6) + 8p(t,15,4) + 0.01,
\end{aligned}$$

where the exponential "peaks" $p(t,a,b) = e^{-b(t-a)^2}$. The system of differential equations is stiff with eigenvalues $\{-10, -2, -3 \pm 5i\}$. Notice also that the single state variable inequality path constraint is defined in terms of the peak functions. The system has boundary conditions

$$\begin{aligned}
\mathbf{y}^{\mathsf{T}}(0) &= [2,1,2,1], \\
\mathbf{y}^{\mathsf{T}}(20) &= [2,3,1,-2]
\end{aligned}$$

and the goal is to minimize the objective function

$$J(\mathbf{y},\mathbf{u}) = \int_0^{20} 10^2(y_1^2 + y_2^2 + y_3^2 + y_4^2) + 10^{-2}(u_1^2 + u_2^2)dt.$$

We refer to this as the Alp rider problem because the minimum value for the objective function tends to force the state to ride the path constraint. Figure 4.6 illustrates the peaks for the path-constraint function for this problem.

In fact, this problem is very similar to the path encountered by a terrain-following aircraft. This is quite obvious in the solution illustrated in Figure 4.7, which demonstrates peaks at the locations $(3,6,10,15)$.

The Alp rider example required 11 mesh-refinement iterations, which are tabulated in the rows of Table 4.2. For the results in this table, we initiated the algorithm with

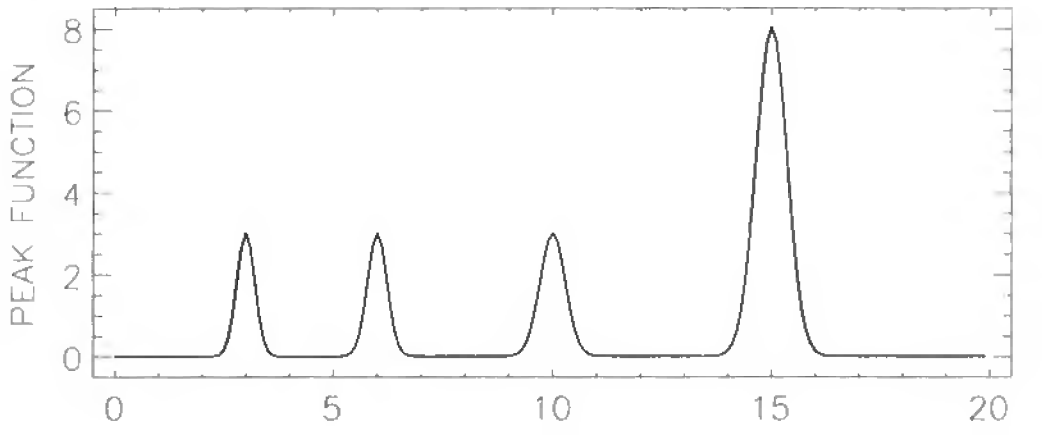


Figure 4.6: Alp rider peaks.

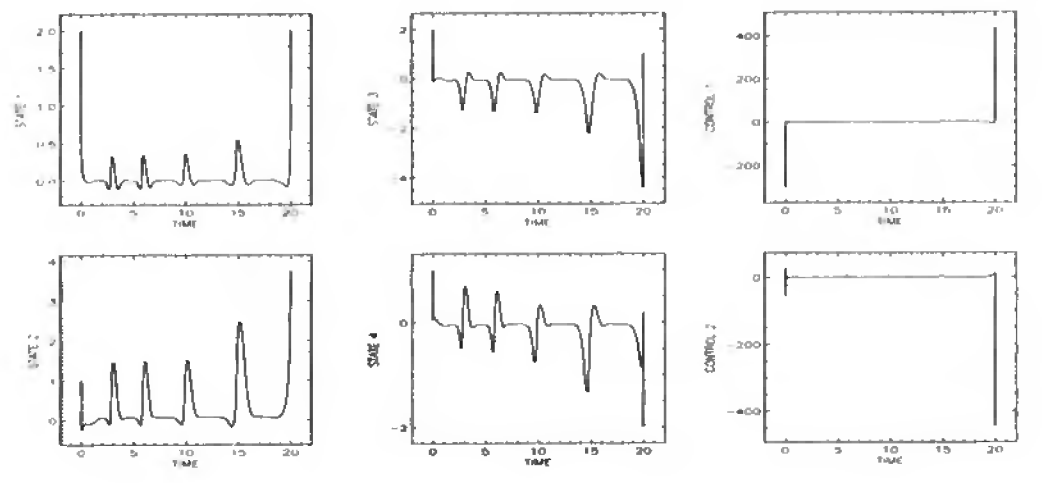


Figure 4.7: Alp rider solution.

GRID	NPT	NGC	NHC	NFE	NRHS	ERRODE	CPU (sec)
1	21	66	62	7529	308689	0.32×10^{0}	0.22×10^{2}
2	41	93	91	10969	888489	0.29×10^{-1}	0.66×10^{2}
3	76	37	35	4244	640844	0.74×10^{-2}	0.63×10^{2}
4	84	19	16	1993	332831	0.89×10^{-3}	0.35×10^{2}
5	119	26	23	2833	671421	0.18×10^{-3}	0.65×10^{2}
6	194	18	16	1964	760068	0.31×10^{-4}	0.95×10^{2}
7	253	17	15	1844	931220	0.11×10^{-4}	0.14×10^{3}
8	304	10	8	1004	609428	0.37×10^{-5}	0.16×10^{3}
9	607	6	4	524	635612	0.35×10^{-6}	0.54×10^{3}
10	785	5	3	404	633876	0.16×10^{-6}	0.92×10^{3}
11	992	5	3	404	801132	0.29×10^{-7}	0.14×10^{4}
	992	302	276	33712	7213610		3543.81

Table 4.2: Alp rider performance summary.

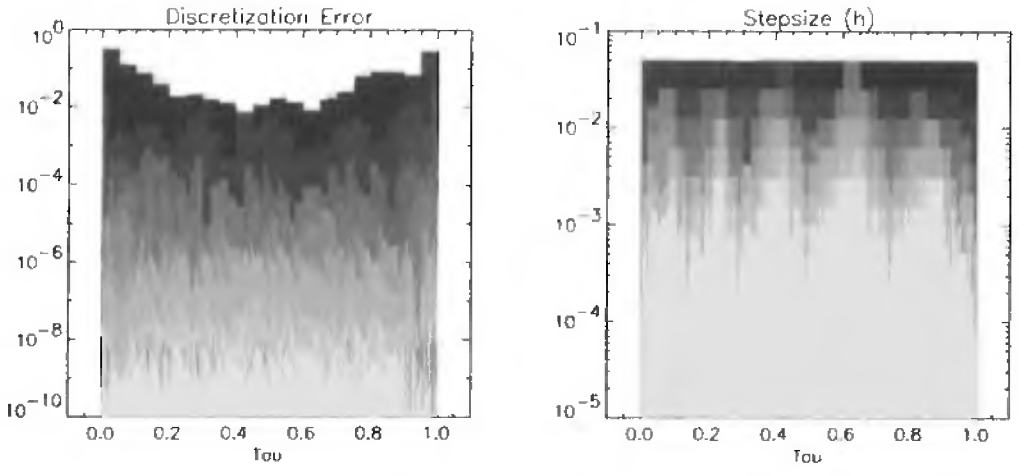


Figure 4.8: Alp rider mesh-refinement history.

the Hermite-Simpson discretization, although results for the default method are very similar. The first iteration began with 21 equally spaced grid points (NPT) and was solved after 66 gradient evaluations (NGC) and 62 Hessian evaluations (NHC) for a total of 7529 function evaluations (NFE), including finite difference perturbations. This solution required 308689 evaluations of the right-hand sides (NRHS) of the DAEs and produced a solution with a discretization error (ERRODE) of 0.32×10^{0}, which was obtained in 0.22×10^{2} sec. Using the solution from the first iteration as an initial guess, the second solution, using 41 grid points, was obtained after an additional 91 Hessian evaluations. Notice that it is necessary to substantially increase the size of the mesh before the solution of the NLP problem is obtained quickly: this suggests that the quadratic convergence region for this application is very small. For comparison, the old refinement method [26] required 27370.44 sec, 2494 grid points, and 10 iterations to solve this problem. This substantial difference in computation time can be attributed to the size of the final mesh. In particular, the computational cost of the NLP problem is related to the number of grid points and, since the final grid for the old method is approximately 2.5 times as large, it is not surprising that the computational cost for the old method is 7.7 times larger than for the new approach.

Figure 4.8 illustrates the behavior of the refinement algorithm on the Alp rider example. The first iteration is shown with the darkest shading and the last iteration has the lightest shading. Observe that the early refinement steps tend to cluster points near the boundaries and peaks where the calculated discretization error is largest. The final mesh-refinement iterations tend to have a more uniform distribution of error because the grid points have been clustered in the appropriate regions.

4.8 Scaling

In Section 1.14, we discussed the importance of scaling an NLP problem in order to obtain robust and rapid convergence to a solution. The special structure of the optimal control problem can be exploited to improve the scaling of the underlying NLP subproblem. Many of the trajectory problems (e.g., Example 5.1) discussed in the next chapter are

formulated using quantities with significant differences in the units for the state variables. For example, some of the state variables represent angles ranging from $-\pi$ to π, where others, like altitude, may range from 0 to 10^6 ft. Thus, scaling is necessary to make the ranges of the variables more uniform. While no scaling method is universally successful, the technique presented is often helpful.

To make the analysis physically meaningful, we would like to choose scaling in the control setting rather than the NLP setting. In other words, we would like to choose scaling for the state variables and the control variables. Thus, for the purposes of this analysis, let us assume that the scaling is the same over all grid points and temporarily drop the grid-point notation. Rewriting the NLP variable scaling (1.124) in vector form and then applying it to the control setting, we find

$$\begin{bmatrix} \tilde{\mathbf{y}} \\ \tilde{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{V}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_u \end{bmatrix} \begin{bmatrix} \mathbf{y} \\ \mathbf{u} \end{bmatrix} + \begin{bmatrix} \mathbf{r}_y \\ \mathbf{r}_u \end{bmatrix} \tag{4.173}$$

which relates the scaled state $\tilde{\mathbf{y}}$ and control $\tilde{\mathbf{u}}$ to the unscaled quantities $\mathbf{y}$ and $\mathbf{u}$. The $n_y \times n_y$ diagonal matrix $\mathbf{V}_y$ contains the state variable scale weights and the $n_u \times n_u$ diagonal matrix $\mathbf{V}_u$ contains the control variable scale weights. The corresponding variable shifts are defined by the vectors $\mathbf{r}_y$ and $\mathbf{r}_u$.

In general, we impose defect constraints $\boldsymbol{\zeta} = \mathbf{0}$ and path constraints $\mathbf{g} = \mathbf{0}$. Thus, from (1.125) we expect

$$\tilde{\mathbf{c}} = \begin{bmatrix} \mathbf{W}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_g \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta} \\ \mathbf{g} \end{bmatrix} \tag{4.174}$$

will relate the scaled constraints $\tilde{\mathbf{c}}$ to the unscaled constraints $\boldsymbol{\zeta}$ and $\mathbf{g}$. Here $\mathbf{W}_f$ is an $n_y \times n_y$ diagonal matrix of differential equation constraint scale weights and $\mathbf{W}_g$ is an $n_g \times n_g$ diagonal matrix of path-constraint scale weights.

The Jacobian matrix in the scaled quantities is given by

$$\tilde{\mathbf{G}} = \begin{bmatrix} \mathbf{W}_f & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_g \end{bmatrix} \mathbf{G} \begin{bmatrix} \mathbf{V}_y & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_u \end{bmatrix}^{-1} \tag{4.175}$$

Now all of the transcription methods (4.49), (4.51), (4.57), and (4.59) are of the form

$$\boldsymbol{\zeta} \sim \mathbf{y} - h_k \mathbf{f}. \tag{4.176}$$

Thus, the unscaled Jacobian matrix has the form

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_f \\ \mathbf{G}_g \end{bmatrix} \sim \begin{bmatrix} \mathbf{I} - h_k \dfrac{\partial \mathbf{f}}{\partial \mathbf{y}} & -h_k \dfrac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \dfrac{\partial \mathbf{g}}{\partial \mathbf{y}} & \dfrac{\partial \mathbf{g}}{\partial \mathbf{u}} \end{bmatrix} \tag{4.177}$$

But notice that as $h_k \to 0$, the rows of the Jacobian corresponding to the differential equations $\mathbf{G}_f \sim \mathbf{I}$. This implies that the conditioning of the Jacobian improves as the stepsize is reduced. Furthermore, if we want to have the scaled Jacobian $\tilde{\mathbf{G}}_f \sim \mathbf{I}$, we can set

$$\mathbf{W}_f = \mathbf{V}_y \tag{4.178}$$

in (4.175). In other words, if the state variable y_k is to be scaled by the weight v_k, then the constraint ζ_k should also be scaled by the same weight. To achieve similar results

for the path constraints, the matrix $\mathbf{W}_g$ should be chosen so that the rows of $\mathbf{G}_g$ have norm one. In other words, it may be desirable to replace the original path constraints with a scaled expression of the form

$$\mathbf{0} = \mathbf{W}_g \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]. \tag{4.179}$$

It is worth noting that the matrix $\mathbf{G}$ (4.177) is closely related to the *iteration matrix* used in the corrector iterations of many numerical integrators [30].

It now remains to choose the variable scaling to deal with the problem of units described above. Ideally, we would like to choose the variable scales $\mathbf{V}$ and shifts $\mathbf{r}$ such that the scaled quantities $\widetilde{\mathbf{y}} \sim \mathcal{O}(1)$ and lie in the interval $[-0.5, +0.5]$. When simple bounds such as (4.35) are available, it is clear that

$$v_k = \frac{1}{y_{U,k} - y_{L,k}}, \tag{4.180}$$

$$r_k = \frac{1}{2} - \frac{y_{U,k}}{y_{U,k} - y_{L,k}}. \tag{4.181}$$

When simple bounds are not part of the problem description, it is necessary to construct equivalent information about the variable ranges. In the SOCS software, this is achieved by examining the user-supplied initial guess.

4.9 Quadrature Equations

The general formulation of an optimal control problem may involve *quadrature functions* as introduced in (4.38). The quadrature functions may appear either in the objective function as in (4.41) or as integral constraints of the form

$$\psi_L \leq \int_{t_I}^{t_F} w\left[\mathbf{y}(t), \mathbf{u}(t), \mathbf{p}, t\right] dt \leq \psi_U. \tag{4.182}$$

To simplify the discussion, let us focus on an optimal control problem stated in Lagrange form, although all results are directly applicable when the integrated quantities are constraints. Recall that the Lagrange problem

$$\min_u J = \int_{t_I}^{t_F} w(\mathbf{y}, \mathbf{u}, t) dt, \tag{4.183}$$

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t). \tag{4.184}$$

$$\mathbf{y}^{\mathsf{T}}(t_I) = \boldsymbol{\psi}_0^{\mathsf{T}} \tag{4.185}$$

can be restated as a Mayer problem of the form

$$\min_u J = y_{n+1}(t_F), \tag{4.186}$$

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \mathbf{u}, t), \tag{4.187}$$

$$\dot{y}_{n+1} = w(\mathbf{y}, \mathbf{u}, t), \tag{4.188}$$

$$\mathbf{y}^{\mathsf{T}}(t_I) = \boldsymbol{\psi}_0^{\mathsf{T}}, \tag{4.189}$$

$$y_{n+1}(t_I) = 0. \tag{4.190}$$

Clearly, the Mayer and Lagrange forms are mathematically equivalent, but are they computationally comparable? The answer is no!

First, the *number* of state variables for the Mayer formulation is greater than the number for the Lagrange formulation. Thus, when the state variable is discretized, the number of NLP variables will be increased by Mn_w, where n_w is the number of quadrature functions and M is the number of grid points. Since the size of the NLP is larger, one can expect some degradation in the computation time relative to the Lagrange form.

The second (less obvious) reason concerns *robustness*. When the Lagrange formulation is discretized, the original ODEs are approximated by defect constraints, $\boldsymbol{\zeta}_k = \mathbf{0}$, such as (4.57) and (4.59). In contrast, when the Mayer formulation is discretized, defect constraints are introduced for *both* the original ODEs (4.187) *and* the quadrature differential equation (4.188). Thus, it may be more difficult for the NLP algorithm to solve both the original ODE defects and the quadrature defect constraints. This effect is most noticeable when the quadrature functions are nonlinear and/or the grid is coarse. In essence, the discretized version of the Lagrange problem is more "forgiving" than the discretized version of the Mayer problem.

From a numerical standpoint, the key notion is to *implicitly* introduce the additional state variable y_{n+1} without actually having it appear explicitly in the relevant expressions. To see how this is achieved, let us construct the quadrature approximation when a trapezoidal discretization is employed:

$$\int_{t_I}^{t_F} w(\mathbf{y}, \mathbf{u}, t)dt = y_{n+1}(t_M) \tag{4.191}$$

$$= \sum_{k=1}^{M-1} \frac{\tau_k \Delta t}{2}[w_{k+1} + w_k]. \tag{4.192}$$

Expression (4.192) is obtained by recursive application of the trapezoidal discretization, e.g., (4.69), with the specified initial condition $y_{n+1}(t_I) = 0$. The final expression can be rewritten as

$$\int_{t_I}^{t_F} w(\mathbf{y}, \mathbf{u}, t)dt = \mathbf{b}^\mathsf{T}\mathbf{q}, \tag{4.193}$$

where the coefficient vector is

$$\mathbf{b}^\mathsf{T} = \frac{1}{2}[\tau_1, (\tau_2 + \tau_1), (\tau_3 + \tau_2), \ldots, (\tau_{M-1} + \tau_{M-2}), \tau_{M-1}] \tag{4.194}$$

and the vector $\mathbf{q}$ has elements

$$q_k = \Delta t\, w_k \tag{4.195}$$

for $k = 1, \ldots, M$. Observe that the elements of $\mathbf{q}$ are functions of the original state and control. Furthermore, the objective function can be computed from the original state and control—y_{n+1} does not appear explicitly in any expression. Thus, we have constructed the objective function in the form (4.115), where the last row of the matrix $\mathbf{B}$ is the vector $\mathbf{b}^\mathsf{T}$. In summary, all of the techniques described in Section 4.6 can be used to construct the NLP Jacobian and Hessian matrices while exploiting sparsity in the quadrature functions w. It should also be clear that the same approach can be used when the quadrature expressions are computed using a Simpson discretization.

The treatment of quadrature equations must also be addressed in the mesh-refinement process. Again, the goal is to *implicitly* form the relevant information without explicitly introducing an additional state variable. If a Mayer formulation was used and an additional state was introduced, there would be a contribution to the discretization error from (4.156) of the form

$$\varepsilon(t) = \dot{\tilde{y}}_{n+1}(t) - w[\tilde{\mathbf{y}}(t), \tilde{\mathbf{u}}(t), t], \tag{4.196}$$

which must be evaluated in order to compute the integral (4.155). To eliminate the explicit contribution of the additional state, we must construct $\dot{\tilde{y}}_{n+1}(t)$. This quantity can be easily computed by simply interpolating the local information. When a trapezoidal quadrature is used, a linear interpolant can be constructed through the values at the nearest grid points, $y_{n+1}(t_k)$ and $y_{n+1}(t_{k+1})$. Similarly, a cubic (Hermite) interpolant can be constructed by also using the values of $w_{n+1}(t_k)$ and $w_{n+1}(t_{k+1})$. In either case, this local interpolating function provides the necessary information to evaluate the discretization error.

The treatment of quadrature equations that has been described

1. does not introduce an additional state variable $y_{n+1}(t)$,
2. does not introduce additional defect constraints,
3. but does adjust the mesh as though the additional state was introduced.

In essence the technique ensures that the discretization mesh is constructed such that all continuous functions (4.39) are accurately represented. In fact we use the same approach to guarantee accuracy in the algebraic equations (4.34).

Example 4.4. Rao and Mease [92] present an example that illustrates the importance of this approach. Rao and Mease refer to this as a "hyper-sensitive" problem. It is extremely difficult to solve using an indirect method, and is equally difficult when treated in the Mayer form. The problem is defined by a single differential equation and is stated in Lagrange form:

$$\min_u J = \int_0^{t_F} \left[y^2 + u^2\right] dt, \tag{4.197}$$

$$\dot{y} = -y^3 + u, \tag{4.198}$$

where $y(0) = 1$, $y(t_F) = 1.5$, and $t_F = 10000$. The state variable history and the corresponding distribution of stepsize are illustrated in Figure 4.9. The initial constant stepsize is shown with a dotted line. Because of the dramatic change in the state variable near the initial and final times, it is necessary to have a very small stepsize in these regions. The SOCS iteration history, beginning with 25 equally spaced grid points, is summarized in Table 4.3. Notice that the discretization error is reduced by 12 orders of magnitude by the mesh-refinement procedure. When the problem is solved in Mayer form, the solution to the very first NLP (with 25 grid points) requires over 1322 iterations and 7860 function evaluations. This is 131 times more than the 60 function evaluations reported in line 1 of Table 4.3.

4.10 What Can Go Wrong

In Section 4.3, we spent some time discussing what's wrong with the indirect method. We are now in a position to consider *what's good about the direct method.*

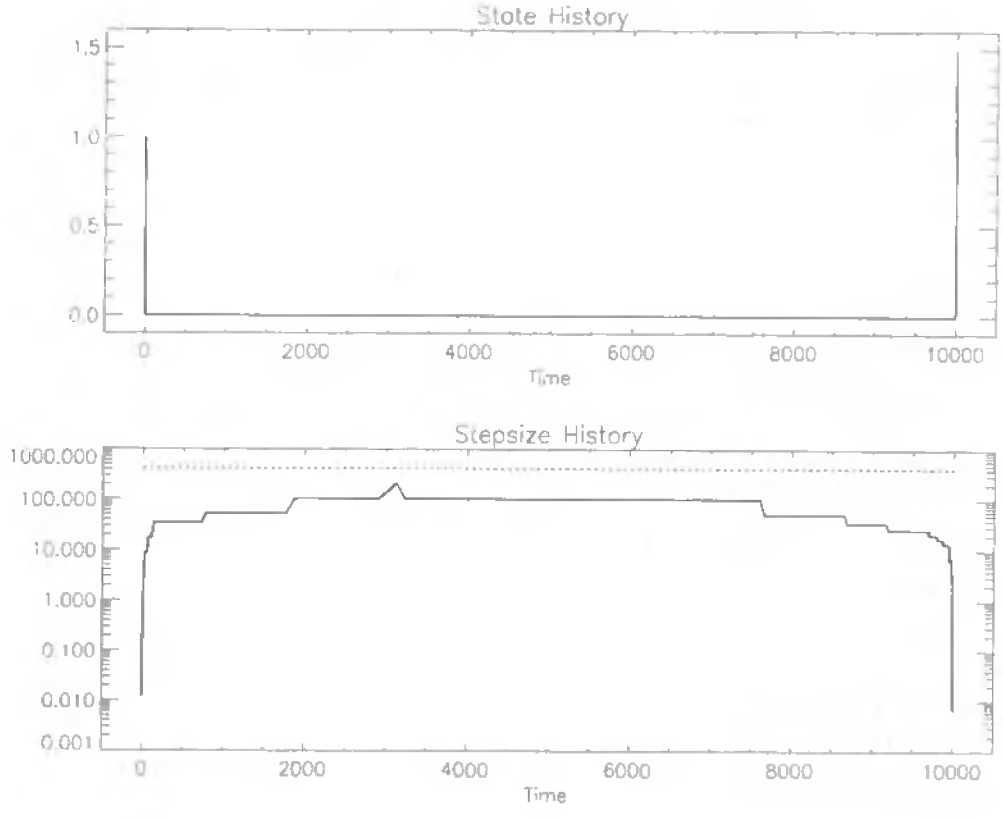


Figure 4.9: Hypersensitive problem.

GRID	NPT	NGC	NHC	NFE	NRHS	ERRODE	CPU (sec)
1	25	14	2	60	1500	0.15×10^{5}	0.62×10^{0}
2	25	21	19	124	6076	0.37×10^{3}	0.15×10^{1}
3	49	11	9	63	6111	0.28×10^{2}	0.18×10^{1}
4	97	11	8	61	11773	0.49×10^{1}	0.29×10^{1}
5	109	29	27	585	126945	0.11×10^{1}	0.58×10^{1}
6	115	10	8	185	42365	0.13×10^{0}	0.25×10^{1}
7	121	8	6	144	34704	0.12×10^{-1}	0.22×10^{1}
8	128	8	1	93	23715	0.96×10^{-3}	0.18×10^{1}
9	137	5	1	60	16380	0.19×10^{-4}	0.15×10^{1}
10	181	4	1	49	17689	0.55×10^{-6}	0.16×10^{1}
11	229	4	1	49	22393	0.31×10^{-7}	0.19×10^{1}
	229	125	83	1473	309651		24.23

Table 4.3: Hypersensitive problem performance summary.

1. Since the adjoint equations are not formed explicitly, analytic derivatives are not required. Instead, equivalent information can be computed using sparse finite differences. Consequently, a user with minimal knowledge of optimal control theory can use the technique! Complex black box applications can be handled easily. The approach is flexible and new formulations are handled readily. Finally, sparsity in the right-hand side of the dynamic equations (4.33) and (4.34) can be treated automatically.

2. Path inequalities (4.34) do not require an a priori estimate of the constrained-arc sequence because the NLP active set procedure automatically determines the arc sequence.

3. The method is very robust since the user only must guess the problem variables $\mathbf{y}, \mathbf{u}$. Furthermore, the NLP globalization strategy, which is designed to improve a *merit function*, has a much larger region of convergence than finding a root of the gradient of the Lagrangian, $\nabla L = 0$, which is the approach used by an indirect method.

For most applications, the direct method is quite powerful and eliminates the deficiencies of an indirect approach. Nevertheless, there are some situations that are problematic, and it is worthwhile to address them.

4.10.1 Singular Arcs

Normally, the optimal control function is uniquely defined by the optimality condition (4.12). On the other hand, a *singular arc* is characterized by having both $\mathbf{H_u} = \mathbf{0}$ *and* having a singular matrix $\mathbf{H_{uu}}$. When this situation appears, the corresponding sparse NLP problem that arises as a part of the direct transcription method is singular. In particular, the projected Hessian matrix is not positive definite. The standard remedial action used by most NLP algorithms is to modify the Hessian matrix used within the NLP problem, for example, using the Levenberg parameter (2.37) described in Section 2.5. While this approach does "fix" the NLP subproblem, it does not address the real difficulty. Conversely, one can attempt to correct the real problem by imposing the necessary condition appropriate for a singular arc, namely

$$\frac{d^2}{dt^2}\left(\mathbf{H_u}\right) = \mathbf{0}. \tag{4.199}$$

If this mixed approach is used with general-purpose software such as SOCS, it is possible to obtain the correct solution. However, this hybrid technique has all of the drawbacks of an indirect method since the analytic necessary conditions must be derived and the arc sequence guessed.

Example 4.5. As an illustration of this situation, consider a simple version of the classical Goddard rocket problem [35]:

$$\begin{aligned} \dot{h} &= v, \\ \dot{v} &= \frac{1}{m}\left[T(t) - \sigma v^2 \exp[-h/h_0]\right] - g, \\ \dot{m} &= -T(t)/c. \end{aligned} \tag{4.200}$$

It is required to find the thrust history $0 \le T(t) \le T_m$ such that the final altitude $h(t_F)$ is maximized for given initial conditions on altitude h, velocity v, and mass m. The problem definition is completed by the following parameters: $T_m = 200$, $g = 32.174$, $\sigma = 5.4915 \times 10^{-5}$, $c = 1580.9425$, $h_0 = 23800.$, $h(0) = v(0) = 0$, and $m(0) = 3$.

It can be demonstrated that the optimal solution consists of three subarcs—the first at maximum thrust $T(t) = T_m$, followed by a singular arc, with a final arc at minimum thrust $T(t) = 0$. On the singular arc, application of (4.199) leads to the nonlinear algebraic constraint

$$0 = T(t) - \sigma v^2 \exp[-h/h_0] - mg$$

$$- \frac{mg}{1 + 4(c/v) + 2(c^2/v^2)} \left[\frac{c^2}{h_0 g} \left(1 + \frac{v}{c} \right) - 1 - 2\frac{c}{v} \right], \qquad (4.201)$$

which uniquely defines the control $T(t)$. Thus, we can formulate the application as a three-phase problem. All three phases must satisfy the differential equations (4.200); however, on phase 2 it is also necessary to satisfy the (index-one) path constraint (4.201). Using this three-phase formulation, the final time for all three phases is also introduced as an NLP variable in the transcribed formulation. By incorporating knowledge of the singular arc, the SOCS software can efficiently and accurately compute the solution. Table 4.4 summarizes the computational performance, and the solution is illustrated by the shaded region in Figure 4.10. Notice that three mesh-refinement iterations were required to reduce the error in the differential equations (ERRODE) by increasing the number of grid points M from 60 to 91. Furthermore, the solution to each NLP subproblem was computed efficiently in terms of the number of function evaluations (NFE) and the CPU time.

Iter.	M	NFE	ERRODE	CPU (sec)
1	60	30	0.67×10^{-4}	0.15×10^{1}
2	79	26	0.55×10^{-5}	0.12×10^{1}
3	91	14	0.70×10^{-7}	0.96×10^{0}
		70		3.64

Table 4.4: Singular arc using hybrid method.

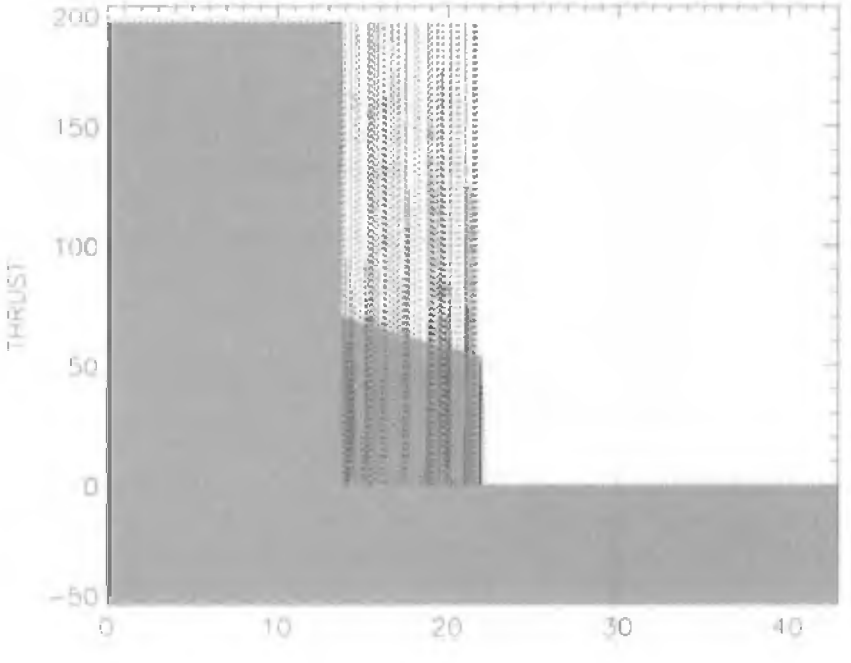


Figure 4.10: Singular arc solution.

Iter	M	NFE	ERRODE	CPU (sec)
1	50	189	0.52×10^{-3}	0.46×10^{1}
2	96	416	0.72×10^{-3}	0.17×10^{2}
3	103	2078	0.13×10^{-3}	0.58×10^{2}
4	188	3398	0.64×10^{-4}	0.34×10^{3}
5	375	230	0.26×10^{-6}	0.19×10^{3}
6	450	230	0.84×10^{-7}	0.17×10^{3}
		6541		775.30

Table 4.5: Singular arc using standard method.

In contrast, when no knowledge of the singular arc is incorporated and a standard one-phase approach is used, the computational performance is much less efficient, as indicated by the results in Table 4.5. Furthermore, even though the final solution had 450 grid points, the control history during the singular arc is quite oscillatory, as illustrated by the dotted line in Figure 4.10. The oscillatory control is, of course, a direct consequence of the fact that the control is not uniquely determined on the singular arc unless the higher-order conditions (4.199) are imposed. The mesh refinement attempts to correct the perceived inaccuracy on the singular arc by adding grid points to this region. It is interesting to note, however, that the computed optimal objective function value $h^* = 18550.6$ is quite close to the true value computed using the hybrid approach ($h^* = 18550.9$).

4.10.2 State Constraints

A standard approach to a problem with path constraint(s)

$$\mathbf{0} \leq \mathbf{g}[\mathbf{y}(t), \mathbf{u}(t), t]$$

is well behaved provided the matrix $\mathbf{g}_u$ is full rank. However, for *state constraints* $\mathbf{g}[\mathbf{y}, \mathbf{u}, t] \equiv \mathbf{g}[\mathbf{y}, t]$, it is clear that the matrix $\mathbf{g}_u$ is *not* full rank. In this case, after discretization, the sparse NLP Jacobian matrix does not have full row rank and, consequently, violates the NLP *constraint-qualification test*! Furthermore, numerically detecting the rank of a matrix is difficult.

On the other hand, if the analyst recognizes this situation, it is possible to use *index reduction* to construct a well-posed discretized subproblem. Essentially the user must *analytically* differentiate $\mathbf{g}$ and then substitute the state equations into the resulting expression. This process can be repeated q times until the control $\mathbf{u}$ appears explicitly. Then, during the constrained arcs, one can impose the derived conditions, e.g.,

$$\frac{d^q}{dt^q}(\mathbf{g}) = \mathbf{0}.$$

Additional endpoint conditions

$$\frac{d^{q-k}}{dt^{q-k}}(\mathbf{g}) \mid_{t=t_a} = \mathbf{0}$$

for $k = 1, \ldots, q$ must be imposed at one end of the constrained arc (where $t = t_a$). For indirect formulations, the adjoint variables also require additional "jump conditions" at

Iter	M	NFE	ERRODE	CPU
1	10	149	0.16×10^{-2}	0.13×10^{1}
2	19	35	0.19×10^{-3}	0.54×10^{0}
3	21	140	0.13×10^{-4}	0.10×10^{1}
4	41	119	0.55×10^{-6}	0.21×10^{1}
5	69	119	0.53×10^{-7}	0.72×10^{1}
		562		12.16

Table 4.6: State-constrained solution using standard method.

the end of the arc. Although this technique can be used to obtain the correct solution, unfortunately it is not "automatic" and, as such, has all of the drawbacks of the indirect method.

Example 4.6. To illustrate the situation, consider the classical Brachistochrone problem with a state variable inequality constraint.

Minimize t_F subject to

$$\begin{aligned} \dot{x} &= v\cos u, \\ \dot{y} &= v\sin u, \\ \dot{v} &= g\sin u, \\ 0 &\geq y - x/2 - h \end{aligned}$$

with boundary conditions $x_0 = y_0 = v_0 = 0$ and $x_F = 1$. Now for $h = 0.1$, the behavior of the standard approach summarized in Table 4.6 efficiently and accurately computes the solution illustrated in Figure 4.11. Unfortunately, using the same technique with $h = 0$ will fail because with $x_0 = 0$, $y_0 = 0$, and $y_0 - x_0/2 = 0$, the Jacobian is singular!

4.10.3 Discontinuous Control

Example 4.7. As a final example of the limitations present in general-purpose software such as SOCS, it is worth reviewing the behavior on a problem with discontinuous optimal control. This situation can occur whenever the differential equations are linear, and is readily illustrated with the solution to the following problem:

Minimize t_F subject to

$$\begin{aligned} \dot{x} &= y, \\ \dot{y} &= u, \\ -1 \leq\ & u \leq 1 \end{aligned}$$

with boundary conditions $x_0 = y_0 = 0$, $x_F = 1$, and $y_F = 0$. The exact optimal solution is bang-bang with

$$u^*(t) = \begin{cases} 1 & \text{if } t < 1, \\ -1 & \text{if } t > 1. \end{cases}$$

The continuous control function $u(t)$ approximation to this solution is illustrated in Figure 4.12. While this is a reasonably good approximation (notice the independent

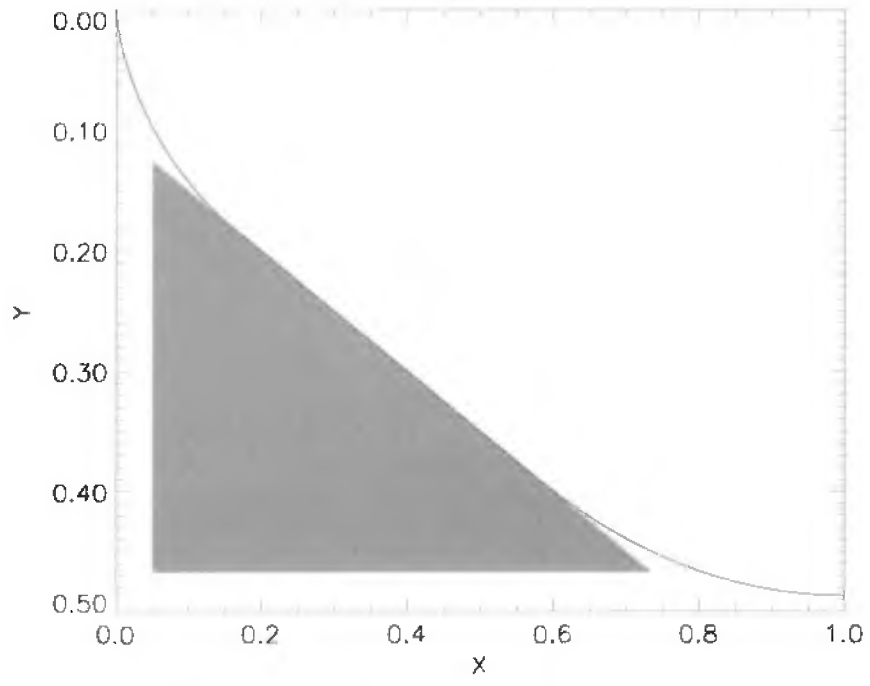


Figure 4.11: State-constrained solution.

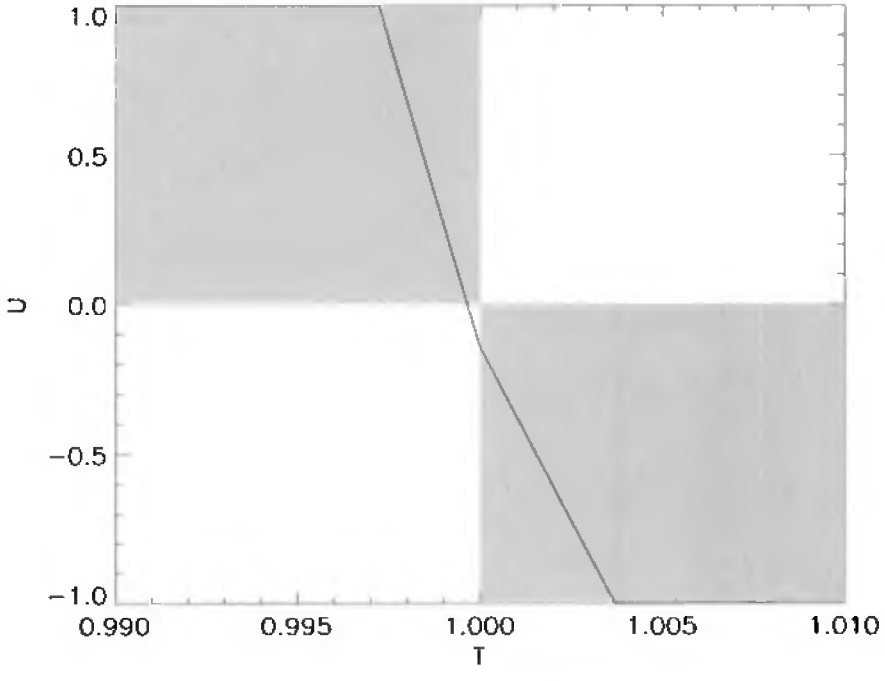


Figure 4.12: Bang-bang control approximation.

variable scale is magnified), it nevertheless is still an approximation. Furthermore, an alternative parameterization of the control that permits discontinuities would readily construct the exact answer. Unfortunately, this again is not "automatic" and, as such, represents a limitation in the current methodology.

Chapter 5

Optimal Control Examples

5.1 Space Shuttle Reentry Trajectory

Construction of the reentry trajectory for the space shuttle is a classic example of an optimal control problem. The problem is of considerable practical interest and is nearly intractable using a simple shooting method because of its nonlinear behavior. Early results were presented by Bulirsch [36] on one version of the problem, as well by Dickmanns [47]. Ascher, Mattheij, and Russell present a similar problem [2, p. 23] and Brenan, Campbell, and Petzold discuss a closely related path control problem [30, p. 157]. Let us consider a particular variant of the problem originally described in [109].

The motion of the vehicle is defined by the following set of DAEs:

$$\dot{h} = v \sin\gamma, \tag{5.1}$$

$$\dot{\phi} = \frac{v}{r} \cos\gamma \sin\psi / \cos\theta, \tag{5.2}$$

$$\dot{\theta} = \frac{v}{r} \cos\gamma \cos\psi, \tag{5.3}$$

$$\dot{v} = -\frac{D}{m} - g \sin\gamma, \tag{5.4}$$

$$\dot{\gamma} = \frac{L}{mv} \cos(\beta) + \cos\gamma \left(\frac{v}{r} - \frac{g}{v} \right), \tag{5.5}$$

$$\dot{\psi} = \frac{1}{mv \cos\gamma} L \sin(\beta) + \frac{v}{r \cos\theta} \cos\gamma \sin\psi \sin\theta, \tag{5.6}$$

$$q \leq q_U, \tag{5.7}$$

where the aerodynamic heating on the vehicle wing leading edge is $q = q_a q_r$ and the dynamic variables are

h	altitude (ft),	γ	flight path angle (rad),
ϕ	longitude (rad),	ψ	azimuth (rad),
θ	latitude (rad),	α	angle of attack (rad),
v	velocity (ft/sec),	β	bank angle (rad).

For the sake of reference, the aerodynamic and atmospheric forces on the vehicle are

specified by the following quantities (English units):

$$
\begin{aligned}
D &= \frac{1}{2} c_D S \rho v^2, & a_0 &= -0.20704, \\
L &= \frac{1}{2} c_L S \rho v^2, & a_1 &= 0.029244, \\
g &= \mu / r^2, & \mu &= 0.14076539 \times 10^{17}, \\
r &= R_e + h, & b_0 &= 0.07854, \\
\rho &= \rho_0 \exp[-h/h_r], & b_1 &= -0.61592 \times 10^{-2}, \\
\rho_0 &= 0.002378, & b_2 &= 0.621408 \times 10^{-3}, \\
h_r &= 23800, & q_r &= 17700 \sqrt{\rho} (0.0001 v)^{3.07}, \\
c_L &= a_0 + a_1 \hat{\alpha}, & q_a &= c_0 + c_1 \hat{\alpha} + c_2 \hat{\alpha}^2 + c_3 \hat{\alpha}^3, \\
c_D &= b_0 + b_1 \hat{\alpha} + b_2 \hat{\alpha}^2, & c_0 &= 1.0672181, \\
\hat{\alpha} &= 180 \alpha / \pi, & c_1 &= -0.19213774 \times 10^{-1}, \\
R_e &= 20902900, & c_2 &= 0.21286289 \times 10^{-3}, \\
S &= 2690, & c_3 &= -0.10117249 \times 10^{-5}.
\end{aligned}
$$

The reentry trajectory begins at an altitude where the aerodynamic forces are quite small with the following initial conditions:

$$
\begin{aligned}
h &= 260000 \text{ ft}, & v &= 25600 \text{ ft/sec}, \\
\phi &= 0 \text{ deg}, & \gamma &= -1 \text{ deg}, \\
\theta &= 0 \text{ deg}, & \psi &= 90 \text{ deg}.
\end{aligned}
$$

The final point on the reentry trajectory occurs at the unknown (free) time t_F, at the so-called terminal area energy management (TAEM) interface, which is defined by the conditions

$$
h = 80000 \text{ ft}, \qquad v = 2500 \text{ ft/sec}, \qquad \gamma = -5 \text{ deg}.
$$

To obtain realistic solutions, we also restrict the trajectory by defining the following simple bounds:

$$
\begin{aligned}
0 &\le h, & -89 \text{ deg} &\le \theta \le 89 \text{ deg}, \\
1 &\le v, & -89 \text{ deg} &\le \gamma \le 89 \text{ deg}, \\
-90 \text{ deg} &\le \alpha \le 90 \text{ deg}, & -89 \text{ deg} &\le \beta \le 1 \text{ deg}.
\end{aligned}
$$

Example 5.1. The goal is to choose the control variables $\alpha(t)$ and $\beta(t)$ such that the final cross-range is maximized. There are many ways to define the cross-range, but for this case it is equivalent to maximizing the final latitude

$$
J = \theta(t_F). \tag{5.8}
$$

For comparison, the solution is computed with no limit on the aerodynamic heating, i.e., $q_U = \infty$, and also with an upper bound on the aerodynamic heating of $q_U = 70$ BTU/ft^2/sec. Figure 5.1 illustrates the optimal trajectory when no heating limit is imposed.

Figure 5.1: Max cross-range shuttle reentry.

The time histories for the state are shown in Figure 5.2. with the unconstrained solution shown as a solid line and the constrained solution as a dotted line. All of the angular quantities are given in degrees, the altitude in multiples of 10^5 ft, velocity in multiples of 10^4 ft/sec, and time in seconds. In Figure 5.3, the control time histories, as well as the aerodynamic heating, are illustrated. Note that for clarity the two solutions for angle of attack are plotted with different scales – the scale corresponding to the unconstrained solution is given on the right side of the figure. The optimal values for the final time and latitude are summarized in Table 5.1. For the heat-constrained example, Figure 5.4 illustrates the behavior of the SOCS mesh-refinement algorithm as the discretization error is reduced below the requested tolerance of 10^{-7}. The first refinement iteration has the darkest shading and the last refinement has the lightest shading. For this case. using a linear initial guess between the boundary conditions, the solution was computed after 10 mesh-refinement iterations.

It is also interesting to ask whether mesh refinement is necessary. In order to assess the importance of mesh refinement, let us consider the following experiment. Suppose the mesh-refinement procedure is terminated after a specific number of iterations. Call the (prematurely obtained) solution $\widehat{\mathbf{u}}(t)$. This approximate "solution" can be used to propagate the trajectory, i.e., let us integrate the differential equations (5.1)–(5.6)

$$\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, \widehat{\mathbf{u}}, t) \tag{5.9}$$

from $t_I = 0$ to t_F. Let us assume the initial conditions are satisfied and assess the relative error in the terminal conditions, i.e.,

$$\epsilon = 100 \max_{k=1}^{n} \left[\frac{|\hat{y}_k(t_F) - y_k(t_F)|}{\max(|\hat{y}_k(t_F)|, |y_k(t_F)|)} \right]. \tag{5.10}$$

In order to obtain an accurate solution of the IVP, we can use any sophisticated numerical integration software to compute the value of $\widehat{\mathbf{y}}(t_F)$. We have chosen to use a variable-order, variable-stepsize Gear [57] integrator with a relative error tolerance of 10^{-14}. In essence, we are trying to assess how well the approximate solution $\widehat{\mathbf{u}}(t)$ to the BVP satisfies the corresponding IVP. Figure 5.5 summarizes the results of this experiment. Observe that when the approximate solution $\widehat{\mathbf{u}}(t)$ corresponds to the result after the

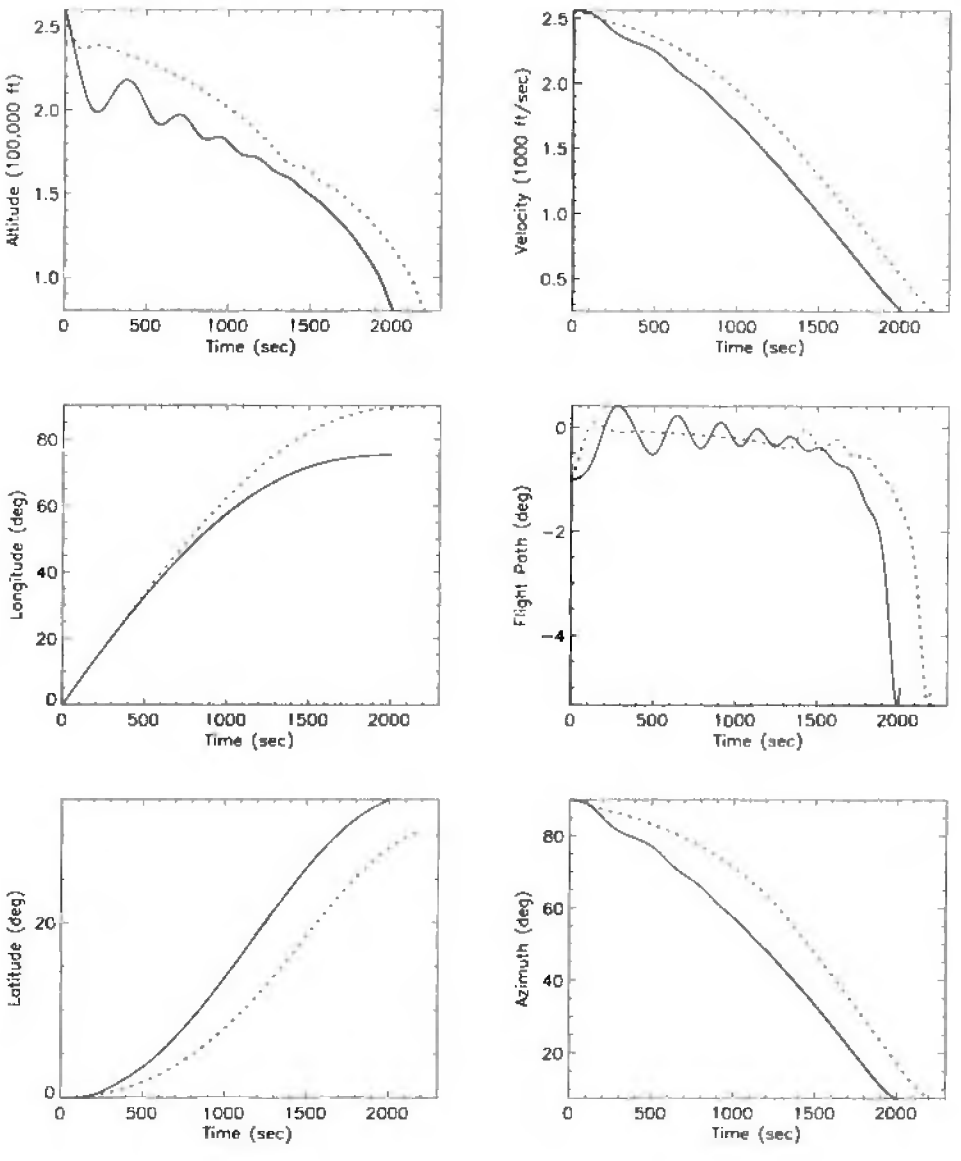


Figure 5.2: Shuttle reentry—state variables.

first mesh-refinement iteration, the relative error in the integrated state is 46.5%. Thus, while the error in the objective function is only 0.4% (which might be reasonable for engineering purposes), the computed trajectory is totally unrealistic. In fact the final integrated trajectory with the approximate control $\widehat{\mathbf{u}}(t)$ has a position error of 12948.73 ft and the velocity error is 390.6111 ft/sec.

Example 5.2. It is interesting to note that the dynamics for Example 5.1 are independent of the longitude ϕ. Observe that the longitude does not appear on the right-hand side of any of the differential equations (5.1)–(5.6). Physically this is not surprising since

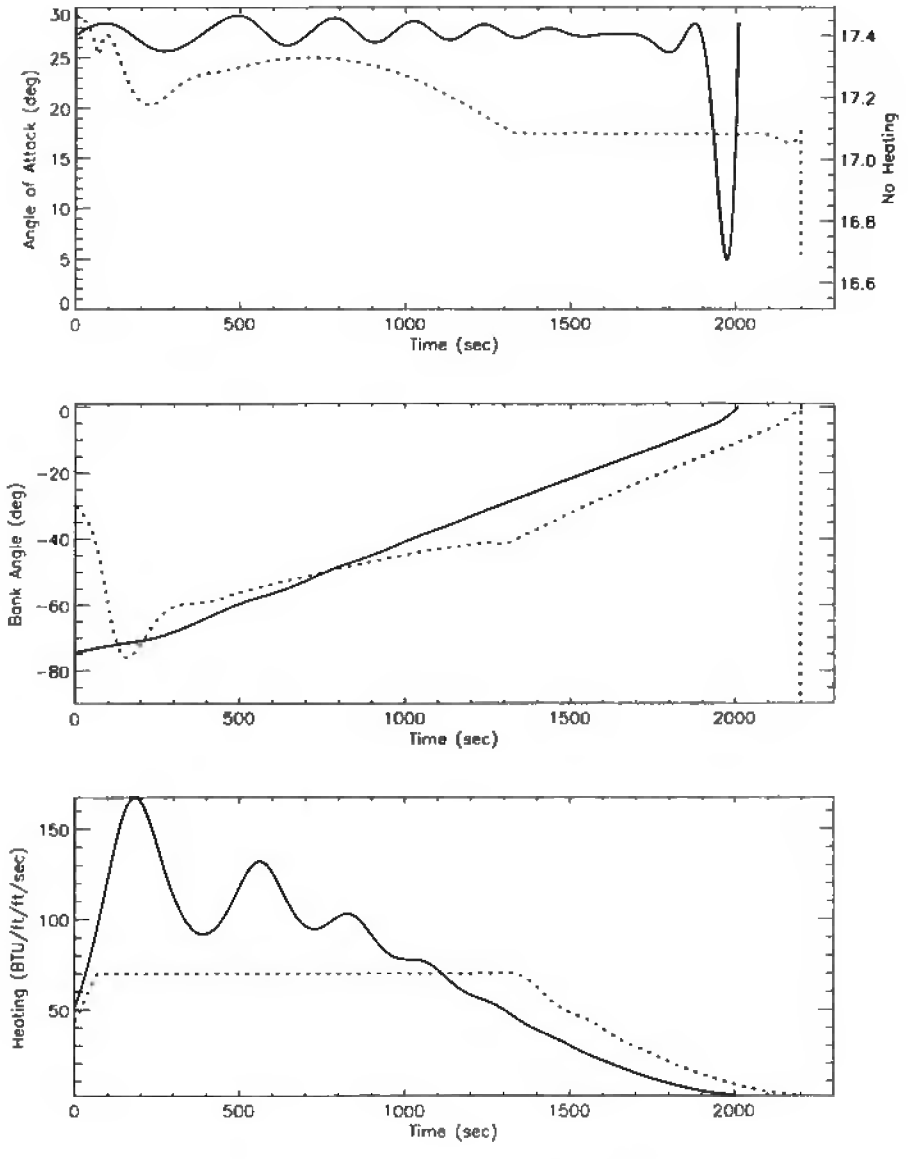


Figure 5.3: Shuttle reentry—control variables.

q_U	∞	70
t_F (sec)	2008.59	2198.67
$\theta(t_F)$ (deg)	34.1412	30.6255

Table 5.1: Shuttle reentry example.

a spherical potential model is used to describe the earth. For convenience, in Example 5.1 the initial longitude was chosen to be $\phi = 0$. Another alternative is to simply eliminate the longitude as a state variable and the corresponding differential equation (5.2). Solutions to the problem using this formulation were presented in Section 2.9.1.

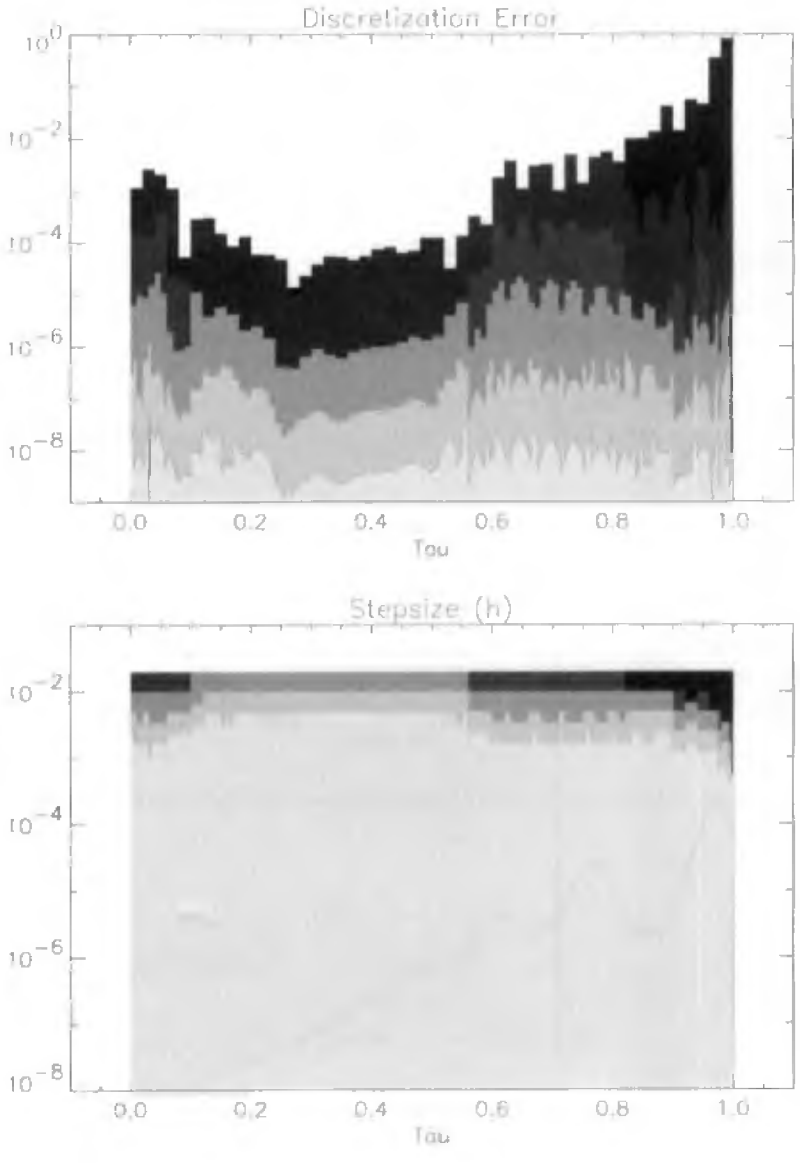


Figure 5.4: Shuttle reentry—mesh refinement.

5.2 Minimum Time to Climb

Example 5.3. The original minimum time to climb problem was presented by Bryson et al. [34] and has been the subject of many analyses since then. Although the problem is not nearly as difficult to solve as the shuttle reentry examples, it is included here because it illustrates the treatment of tabular data. The basic problem is to choose the optimal control function $\alpha(t)$ (the angle of attack) such that an airplane flies from a point on a runway to a specified final altitude as quickly as possible. In its simplest

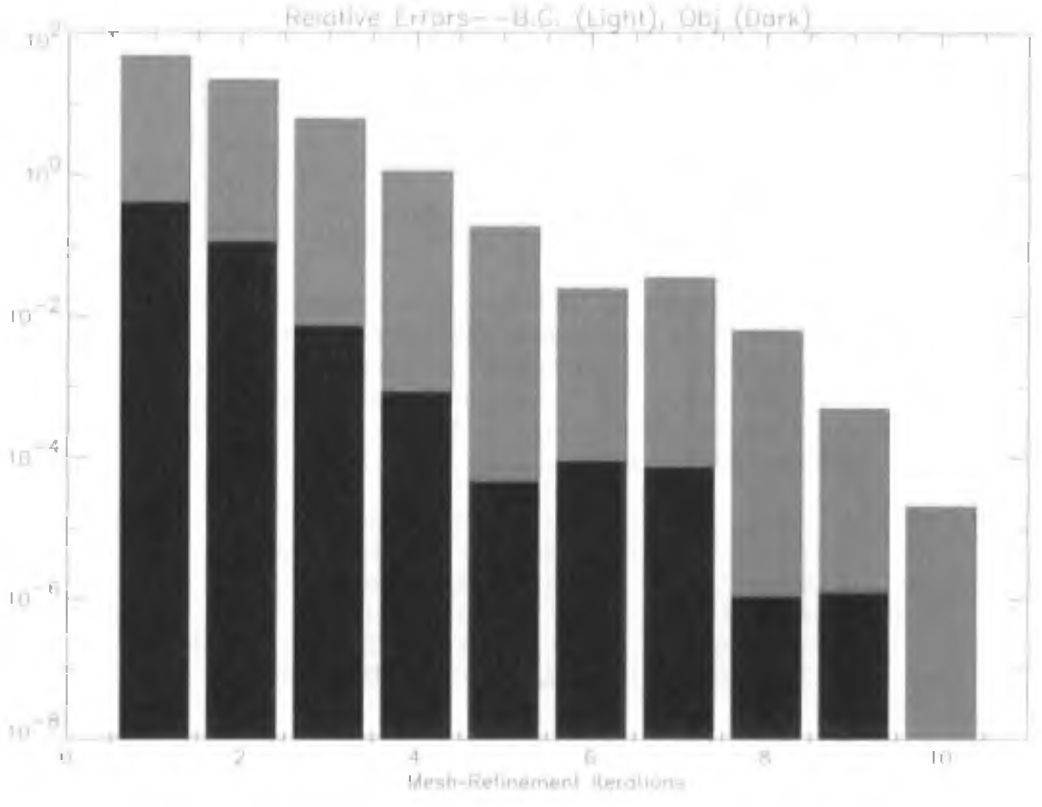


Figure 5.5: Coarse grid solution errors.

form, the planar motion of the aircraft is described by the following set of ODEs:

$$\dot{h} = v \sin\gamma, \tag{5.11}$$

$$\dot{v} = \frac{1}{m}[T(M,h)\cos(\alpha) - D] - \frac{\mu}{(R_e+h)^2}\sin\gamma, \tag{5.12}$$

$$\dot{\gamma} = \frac{1}{mv}[T(M,h)\sin(\alpha) + L] + \cos\gamma\left[\frac{v}{(R_e+h)} - \frac{\mu}{v(R_e+h)^2}\right], \tag{5.13}$$

$$\dot{w} = \frac{-T(M,h)}{I_{sp}}, \tag{5.14}$$

where h is the altitude (ft), v the velocity (ft/sec), γ the flight path angle (rad), w the weight (lb), $m = w/g_0$ the mass, μ the gravitational constant, and R_e the radius of the earth. Furthermore, the simple bounds

$$\begin{aligned} 0 \le h \le 69000.\ (\text{ft}), \qquad & 1 \le v \le 2000.\ (\text{ft}). \\ -89\ (\text{deg}) \le \gamma \le 89\ (\text{deg}), \qquad & 0 \le w \le 45000\ (\text{lb}), \\ -20\ (\text{deg}) \le \alpha \le 20\ (\text{deg}) & \end{aligned}$$

are also imposed.

The aerodynamic forces on the vehicle are defined by the expressions

$$D = \frac{1}{2} C_D S \rho v^2, \tag{5.15}$$

$$L = \frac{1}{2} C_L S \rho v^2, \tag{5.16}$$

$$C_L = c_{L\alpha}(M)\alpha, \tag{5.17}$$

$$C_D = c_{D0}(M) + \eta(M) c_{L\alpha}(M) \alpha^2, \tag{5.18}$$

where D is the drag, L is the lift, C_L and C_D are the aerodynamic lift and drag coefficients, respectively, with S the aerodynamic reference area of the vehicle, and ρ is the atmospheric density. Although the results presented here use a cubic spline approximation to the 1962 Standard Atmosphere, qualitatively similar results can be achieved with a simple exponential approximation to $\rho(h)$ (cf. Example 5.1). The following constants complete the definition of the problem:

$$\begin{array}{llll} h(0) = 0.\ (\text{ft}), & & h(t_F) = 65600.0\ (\text{ft}), \\ v(0) = 424.260\ (\text{ft/sec}), & & v(t_F) = 968.148\ (\text{ft/sec}), \\ \gamma(0) = 0.\ (\text{rad}), & & \gamma(t_F) = 0.\ (\text{rad}), \\ w(0) = 42000.0\ (\text{lb}), & & S = 530.\ (\text{ft}^2), \\ I_{sp} = 1600.0\ (\text{sec}), & & \mu = 0.14076539 \times 10^{17}\ (\text{ft}^3/\text{sec}^2), \\ g_0 = 32.174\ (\text{ft/sec}^2), & & R_e = 20902900.\ (\text{ft}). \end{array}$$

Thrust $T(M, h)$ (thousands of lb)

	Altitude h (thousands of ft)									
M	0	5	10	15	20	25	30	40	50	70
0.0	24.2									
0.2	28.0	24.6	21.1	18.1	15.2	12.8	10.7			
0.4	28.3	25.2	21.9	18.7	15.9	13.4	11.2	7.3	4.4	
0.6	30.8	27.2	23.8	20.5	17.3	14.7	12.3	8.1	4.9	
0.8	34.5	30.3	26.6	23.2	19.8	16.8	14.1	9.4	5.6	1.1
1.0	37.9	34.3	30.4	26.8	23.3	19.8	16.8	11.2	6.8	1.4
1.2	36.1	38.0	34.9	31.3	27.3	23.6	20.1	13.4	8.3	1.7
1.4		36.6	38.5	36.1	31.6	28.1	24.2	16.2	10.0	2.2
1.6				38.7	35.7	32.0	28.1	19.3	11.9	2.9
1.8						34.6	31.1	21.7	13.3	3.1

Table 5.2: Propulsion data.

M	0	0.4	0.8	0.9	1.0	1.2	1.4	1.6	1.8
$c_{L\alpha}$	3.44	3.44	3.44	3.58	4.44	3.44	3.01	2.86	2.44
c_{D0}	0.013	0.013	0.013	0.014	0.031	0.041	0.039	0.036	0.035
η	0.54	0.54	0.54	0.75	0.79	0.78	0.89	0.93	0.93

Table 5.3: Aerodynamic data.

5.2.1 Tabular Data

As with most real aircraft, the aerodynamic and propulsive forces are specified in tabular form. For the sake of completeness, the data as it appeared in the original reference [34] are given in Tables 5.2 and 5.3. There are a number of significant points that characterize the *tabular data* representation. First, both the thrust and aerodynamic table values are

given to limited numeric precision (i.e., approximately two significant figures). Perhaps the most obvious explanation for the limited precision is that the data probably were originally obtained from experimental tests and truncated at the precision of the test equipment. Unfortunately, the statistical analysis (if any) of the original test data has long since been forgotten. Consequently, it is common to assume that the table values are "exact" and correct to full machine precision. A second difficulty, which is evident in the bivariate thrust data, is that apparently data are missing from the corners of the table. Of course, the data are "missing" because a real aircraft can never fly in these regimes (e.g., at Mach number $M = 0$ and $h = 70000$ ft). In fact, for most experimentally obtained data, it can be expected that information will be missing for unrealistic portions of the domain. In view of these realities, it is common to linearly interpolate and never extrapolate the tabular data. Figure 5.6 illustrates a linear treatment of the thrust table.

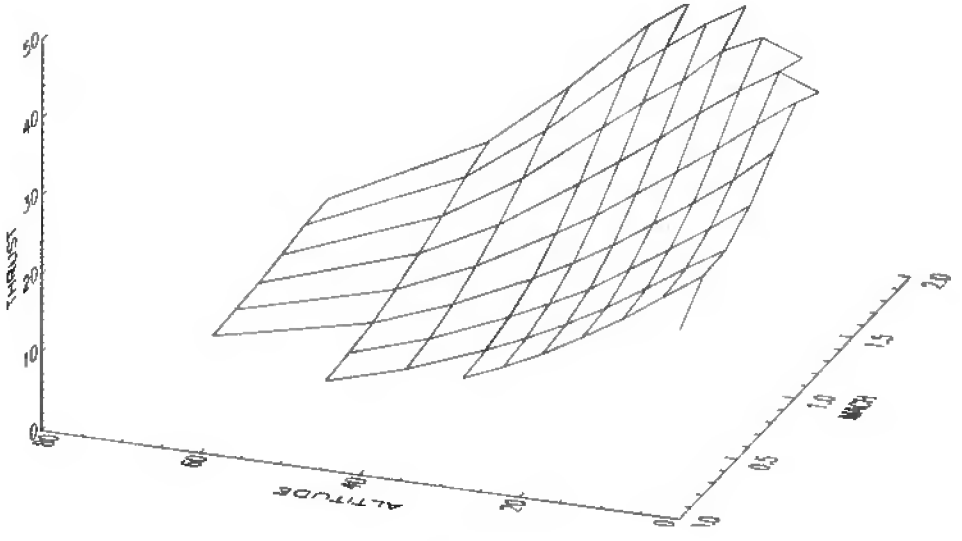


Figure 5.6: Original thrust data.

5.2.2 Cubic Spline Interpolation

While a linear treatment of a tabular function may be adequate for simply evaluating the functions, it is totally inappropriate if the functions are to be used within a trajectory simulation and/or optimization. The principle difficulty (as discussed in Section 1.14) stems from the fact that most numerical optimization and integration algorithms assume that the functions are continuously differentiable to at least second order. Thus, just to propagate the trajectory using an integration algorithm such as Runge–Kutta or Adams Moulton, it is necessary that the right-hand side of the differential equations (5.11)–(5.14) have the necessary continuity. Although it is appealing to ignore a discontinuity, this is usually a poor idea (cf. [68, pp. 196]). Similar requirements are imposed when a numerical optimization algorithm is used to shape the trajectory since the optimization uses second derivative (Hessian) information to construct estimates of the solution.

The most direct way to achieve the required continuity is to approximate the data by a tensor product cubic B-spline of the form

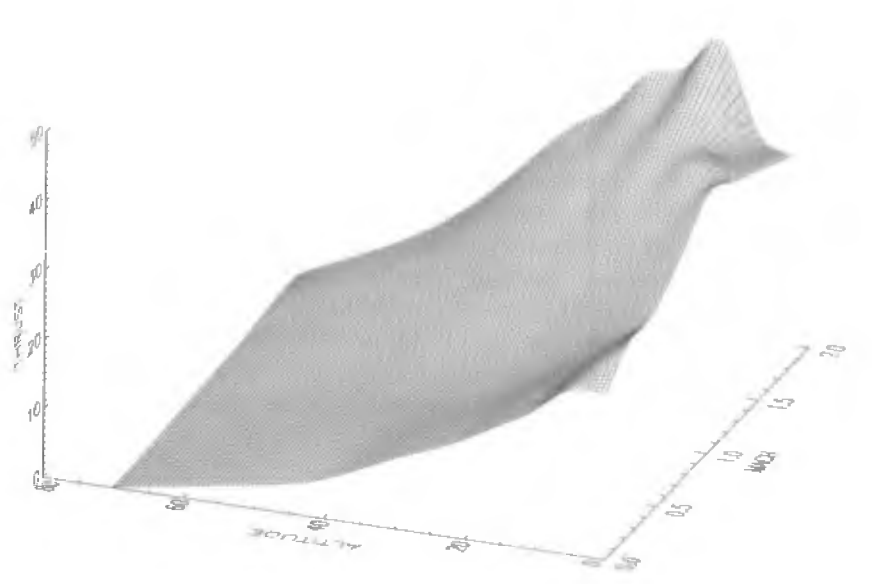


Figure 5.7: Cubic spline interpolant for thrust data.

$$T(M,h) = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} c_{i,j} B_i(M) B_j(h). \tag{5.19}$$

In order to use this approximation, it is necessary to compute the coefficients $c_{i,j}$. The simplest way to compute the spline coefficients is to force the approximating function to *interpolate* the data at all of the data points. However, for a unique interpolant, data are required at all points on a rectangular grid. Since data are missing in the corners of the domain, this difficulty is typically resolved by adding "fake" data in the corners using an "eyeball" approach. It is common to ignore the limited precision of the data and simply treat data as though they were of full precision. Finally, by using divided difference estimates of the derivatives at the boundary of the region, the spline coefficients are uniquely determined by solving a sparse system of linear equations. Figures 5.7 and 5.8 illustrate the cubic interpolating spline obtained using this approach.

5.2.3 Minimum Curvature Spline

The cubic spline interpolant does provide C^2 (second derivative) continuity as needed for proper behavior of integration and optimization algorithms. Unfortunately, the approximation, produced by simply interpolating the raw data, does not necessarily reflect the qualitative aspects of the data. In particular, it is common for the interpolant to introduce "wiggles" that are not actually present in the tabular data itself. This is clearly illustrated along the boundaries of the thrust surface in Figure 5.7 and especially in the aerodynamic data for $M \leq 0.8$ as shown in Figure 5.8. In the case of the latter, it is obvious that the interpolant does not reflect the fact that η is constant for low Mach numbers. A second drawback of the cubic interpolant is the need for data at all points on a rectangular grid when constructing an approximation in more than one dimension.

A technique for eliminating the oscillations in the approximation is to carefully select the spline knot locations by inspection of the data, with fewer knots than data points. In

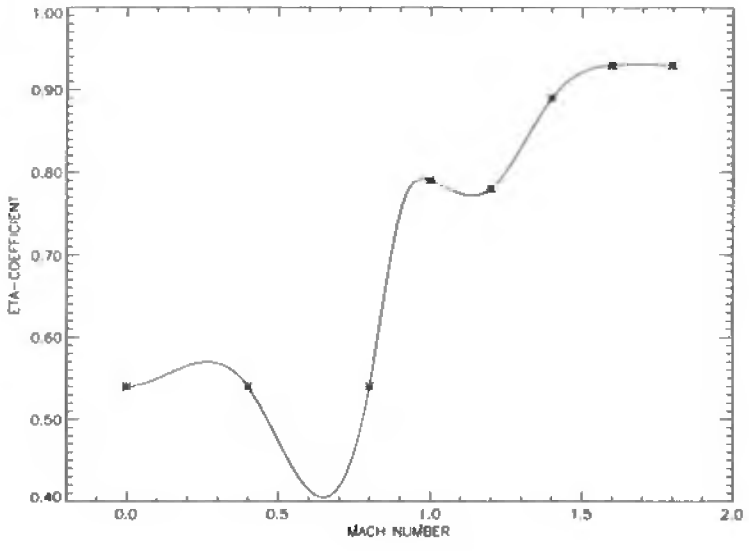


Figure 5.8: Cubic spline interpolant for aerodynamic data.

this case, it is no longer possible to interpolate the data because the number of coefficients is less than the number of interpolation conditions, i.e., the system is overdetermined. However, it is reasonable to determine the spline coefficients $c_{i,j}$ such that

$$f(c_{i,j}) = \sum_{k=1}^{\ell} \left[T(M_k, h_k) - \hat{T}_k \right]^2 \tag{5.20}$$

is minimized. Furthermore, it is possible to introduce constraints on the slope of the spline approximation to reflect monotonicity of data, i.e.,

$$\left[\frac{\partial T}{\partial M} \right] \geq 0 \qquad \text{if } \left[\hat{T}_{k+1} - \hat{T}_k \right] \geq 0, \tag{5.21}$$

$$\left[\frac{\partial T}{\partial M} \right] \leq 0 \qquad \text{if } \left[\hat{T}_{k+1} - \hat{T}_k \right] \leq 0 \tag{5.22}$$

for $M \in [M_k, M_{k+1}]$. Similar constraints can be imposed to reflect monotonicity in the h-direction. The coefficients that satisfy these conditions can be determined by solving a sparse constrained linear least squares problem using the method described in Section 2.10.

By imposing monotonicity constraints and minimizing the error between the data and the approximating function, it is possible to achieve one of the major goals, namely constructing a C^2 function, which eliminates the wiggles. Unfortunately, the location of the knots in the spline approximation must be chosen such that the coefficients are *well* determined by minimizing the least square error. In fact, special care must be taken not to locate knots in regions where data are missing since this will result in a rank-deficient least squares problem. In essence, the knots must be located such that local constraint and data uniquely define the spline coefficients.

To resolve these deficiencies, we introduce a rectangular grid with k_1 values in the first coordinate and k_2 values in the second coordinate. We require that all data points

lie on the rectangular grid. However, not all grid points need have data (i.e., data can be missing). Then let us consider introducing a spline with (a) double knots at the data points in order to ensure C^2 continuity and (b) single knots at the midpoint of each interval. Then let us determine the spline coefficients $c_{i,j}$ that minimize the "curvature"

$$f(c_{i,j}) = \sum_{k=1}^{L} \left[\frac{\partial^2 T}{\partial M^2}(M_k, h_k) \right]^2 + \left[\frac{\partial^2 T}{\partial h^2}(M_k, h_k) \right]^2 \tag{5.23}$$

and satisfy the data approximation constraints

$$\hat{T}_k - \epsilon \leq T(M_k, h_k) \leq \hat{T}_k + \epsilon \tag{5.24}$$

for all data points $k = 1, \ldots, \ell$, where ϵ is the data precision. In order to ensure full rank in the Hessian of the objective, we evaluate the curvature at points determined by the knot-interlacing conditions [44]. We retain the slope constraints (5.21) and (5.22) on the approximation in order to reflect the monotonicity of the data. These coefficients can be determined by solving a sparse constrained linear least squares problem. The resulting approximations are illustrated in Figure 5.9 and Figure 5.10. For this particular fit, the least squares problem had 900 variables with 988 constraints of which 77 were equalities. In general, the number of variables n for a bivariate minimum curvature fit is $n = 9k_1k_2$. For the general case, the minimum curvature formulation requires imposition of m constraints, where $10k_1k_2 \leq m \leq 14k_1k_2$ and the exact number depends on the number of algebraic sign changes in the slope of the data. In addition, if interpolation is required, the number of equality constraints is $m_e \leq k_1k_2$. The entire procedure described for this application has been automated for general multivariate functions with missing data as part of the SOCS software library.

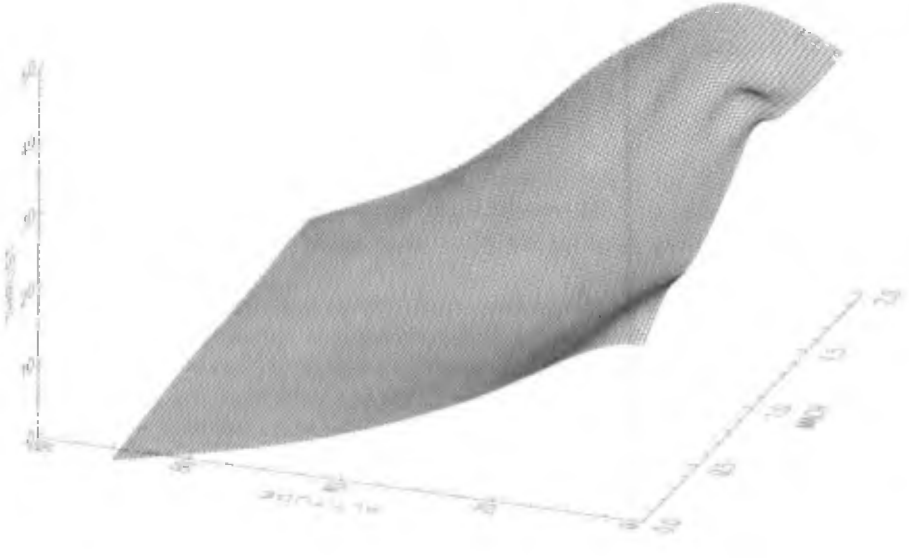


Figure 5.9: Minimum curvature spline for thrust data.

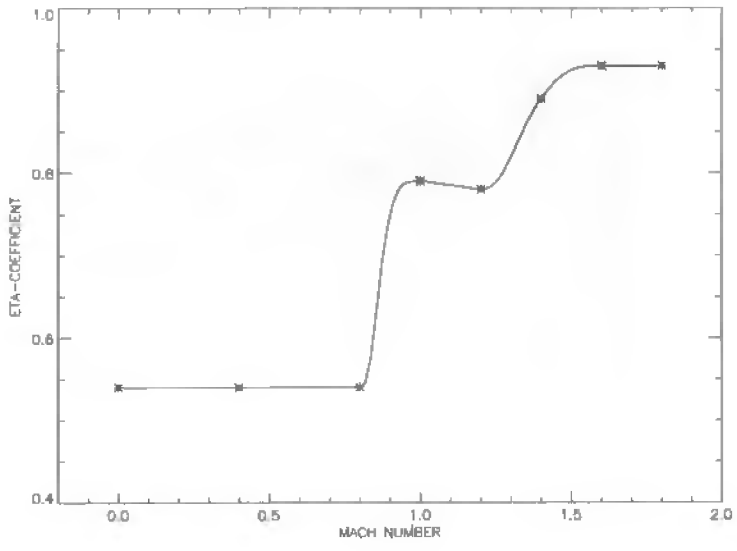


Figure 5.10: Minimum curvature spline for aerodynamic data.

5.2.4 Numerical Solution

Using the minimum curvature approximations for the tabular data, the minimum time to climb problem can be solved using the direct transcription algorithm in SOCS. Table 5.4 summarizes the progress of the algorithm for this application using a linear initial guess for the dynamic variables. The first grid used a trapezoidal discretization (TR) with 10 grid points. The NLP problem was solved using 25 gradient evaluations (GE), 16 Hessian evaluations (HE), and a total of 523 function evaluations (FE) including the finite difference perturbations. The right-hand sides of the ODEs were evaluated 5230 times (NRHS) leading to a solution with a discretization error of $\epsilon_{\max} = 0.35$. Because the error was not sufficiently equidistributed, a second iteration using the trapezoidal discretization was performed. The HSS discretization (HS) was used for the third, fourth, and fifth refinement iterations, after which the HSC method (HC) was used for the remaining refinements. Figure 5.11 illustrates the progress of the mesh-refinement algorithm with the first refinement iteration shaded darkest and the last refinement shaded lightest. For this case, 9 mesh-refinement iterations were required.

Iter.	Disc.	M	GE	HE	FE	NRHS	$\epsilon_{\max}$	CPU (sec)
1	TR	10	25	16	523	5230	0.35×10^{0}	2.8
2	TR	19	8	4	159	3021	0.68×10^{-1}	1.7
3	HS	19	8	5	174	6438	0.87×10^{-2}	3.4
4	HS	37	5	1	74	5402	0.51×10^{-3}	3.7
5	HS	59	4	1	61	7137	0.68×10^{-4}	7.5
6	HC	117	4	1	154	35882	0.12×10^{-4}	11.
7	HC	179	4	1	154	54978	0.13×10^{-5}	16.
8	HC	275	4	1	154	84546	0.14×10^{-6}	27.
9	HC	285	3	1	129	73401	0.97×10^{-7}	22.
Total	-	-	65	31	1582	276035	-	94.88

Table 5.4: Minimum time to climb example.

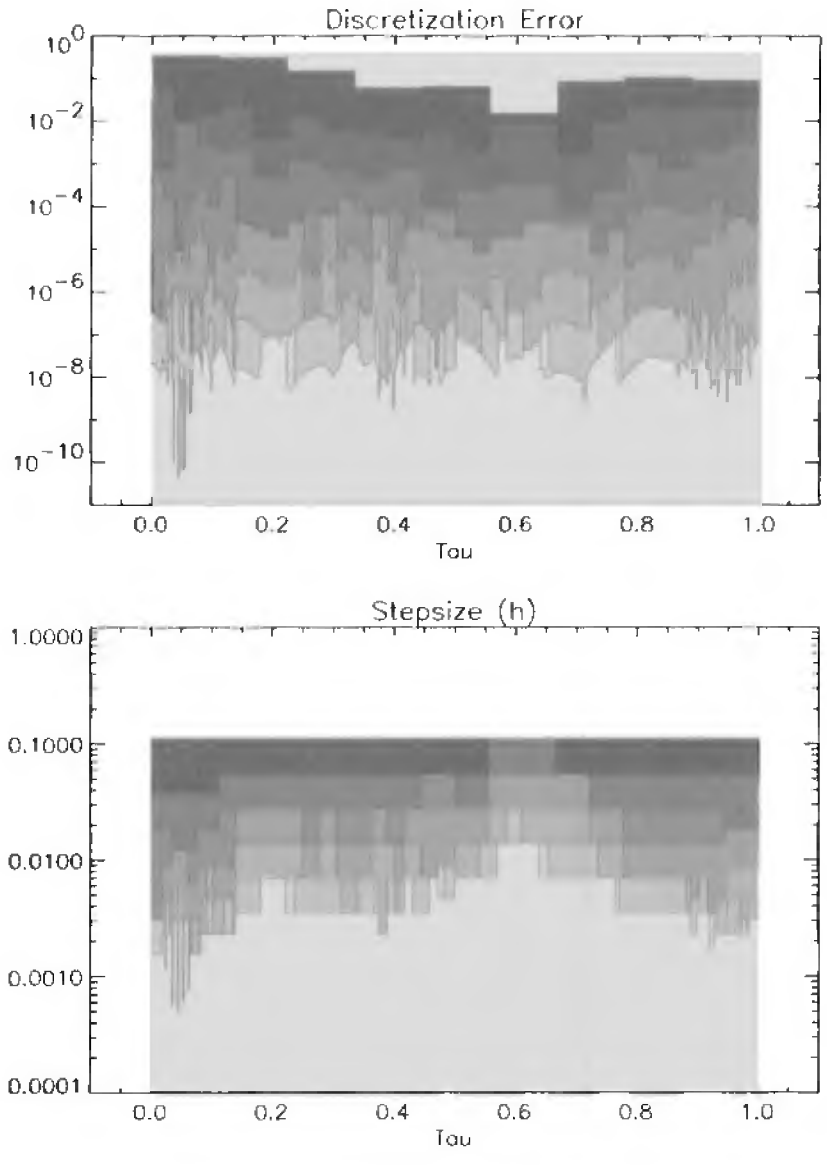


Figure 5.11: Minimum time to climb—mesh refinement.

Figure 5.12 shows the solution with altitude in multiples of 10000 ft, velocity in multiples of 100 ft/sec, and weight in multiples of 10000 lb. The optimal (minimum) time for this trajectory is 324.9750302 (sec). The altitude time history demonstrates one of the more amazing features of the optimal solution, namely the appearance of a *dive* midway through the minimum time to climb trajectory. When first presented in 1969, this unexpected behavior sparked considerable interest and led to the so-called energy-state approach to trajectory analysis. In particular, along the final portion of the trajectory, the energy is nearly constant, as illustrated in the plot of altitude versus velocity.

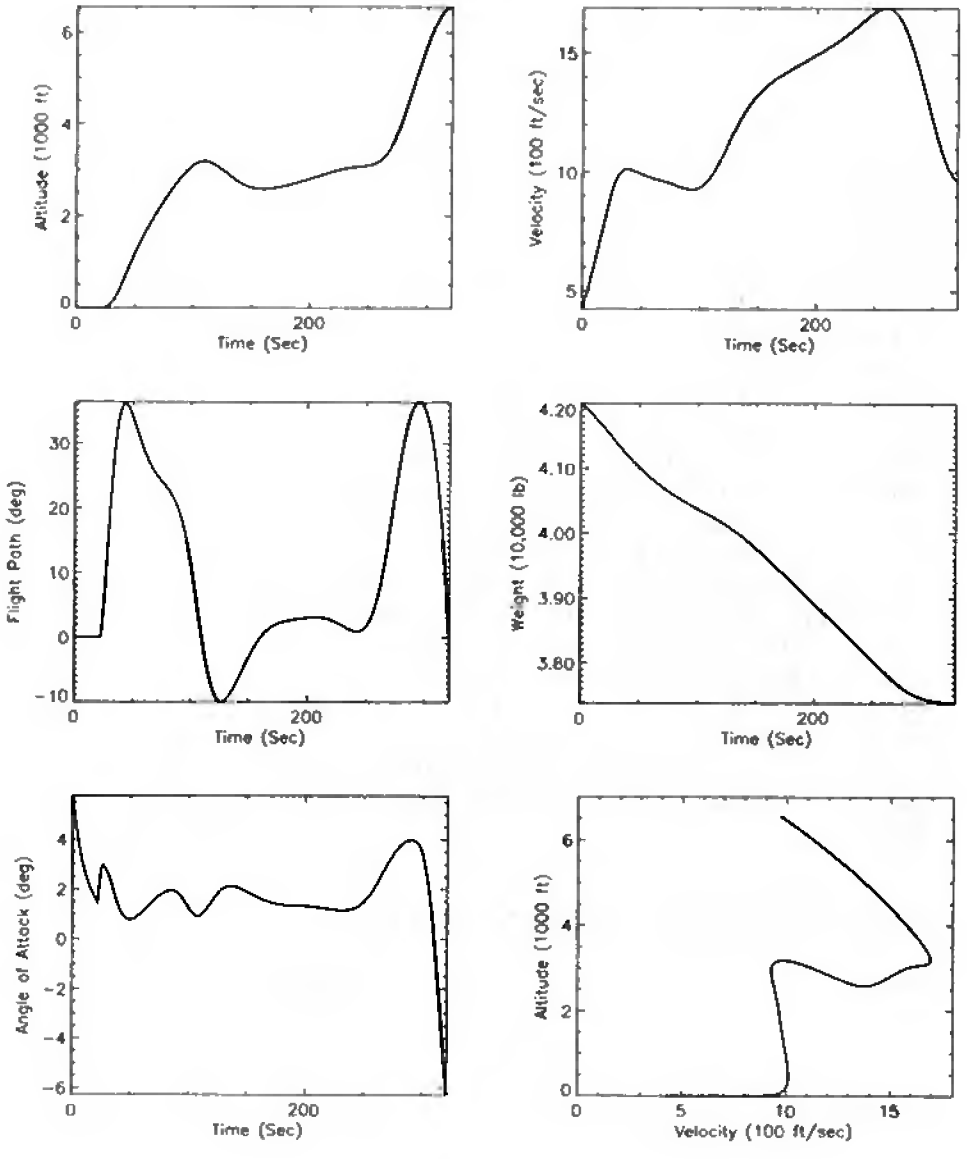


Figure 5.12: Minimum time to climb solution.

5.3 Low-Thrust Orbit Transfer

Example 5.4. Constructing the trajectory for a spacecraft as it transfers from a low earth orbit to a mission orbit leads to a class of challenging optimal control examples. The dynamics are very nonlinear and, because the thrust applied to the vehicle is small in comparison to the weight of the spacecraft, the duration of the trajectory can be very long. Problems of this type have been of considerable interest in the aerospace industry [9, 11, 14, 15, 48, 49, 93, 110]. Typically, the goal is to construct the optimal steering during the transfer such that the final weight is maximized (i.e., minimum fuel

consumed).

The motion of a vehicle can be described by a system of second-order ODEs

$$\ddot{\mathbf{r}} + \mu \frac{\mathbf{r}}{r^3} = \mathbf{a}_d, \tag{5.25}$$

where the radius $r = \|\mathbf{r}\|$ is the magnitude of the inertial position vector $\mathbf{r}$ and μ is the gravitational constant. In this formulation, we define the vector $\mathbf{a}_d$ as the *disturbing acceleration*. This representation for the equations of motion is referred to as Gauss' form of the variational equations.

The Gauss form of the equations of motion isolates the disturbing acceleration from the central force gravitational acceleration. Note that when the disturbing acceleration is zero, $\|\mathbf{a}_d\| = 0$, the fundamental system (5.25) is just a two-body problem. The solution of the two-body problem can, of course, be stated in terms of the constant orbital elements. For low-thrust trajectories, this formulation is appealing because we expect $\|\mathbf{a}_d\|$ to be "small" and, consequently, we expect that the solution can be described in terms of "almost constant" orbital elements. In order to exploit the benefits of the variational form of the differential equations (5.25), it is necessary to transform the Cartesian state into an appropriate set of orbit elements. One potential set contains the classical elements $(a, e, i, \Omega, \omega, M)$. However, these elements exhibit singularities for $e = 0$ and $i = 0$ deg or 90 deg. A set of *equinoctial* orbital elements that avoid the singularities in the classical elements has been described in [5], [31], and [48]. Kechichian developed a particular form of these equations in [79], [78], and [108]. These equations were used to solve a low-thrust earth orbit transfer problem as described in [9]. Unfortunately, this set of equinoctial elements does not accommodate orbits with $e \geq 1$. To eliminate this deficiency, a modified set of equinoctial orbit elements is described in [11] based on the work in [107].

5.3.1 Modified Equinoctial Coordinates

The dynamics of the system can be described in terms of the state variables

$$[\mathbf{y}^\mathsf{T}, w] = [p, f, g, h, k, L, w], \tag{5.26}$$

the control variables

$$\mathbf{u}^\mathsf{T} = [u_r, u_\theta, u_h], \tag{5.27}$$

and the unknown parameter τ.

Using the modified equinoctial elements, the equations of motion for a vehicle with variable thrust can be stated as

$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta} + \mathbf{b}, \tag{5.28}$$

$$\dot{w} = -T\left[1 + 0.01\tau\right]/I_{sp}, \tag{5.29}$$

$$0 = \|\mathbf{u}\| - 1, \tag{5.30}$$

$$\tau_L \leq \tau \leq 0. \tag{5.31}$$

The equinoctial dynamics are defined by the matrix

$$\mathbf{A} = \begin{bmatrix} 0 & \frac{2p}{q}\sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}}\sin L & \sqrt{\frac{p}{\mu}}\frac{1}{q}\{(q+1)\cos L + f\} & -\sqrt{\frac{p}{\mu}}\frac{g}{q}\{h\sin L - k\cos L\} \\ -\sqrt{\frac{p}{\mu}}\cos L & \sqrt{\frac{p}{\mu}}\frac{1}{q}\{(q+1)\sin L + g\} & \sqrt{\frac{p}{\mu}}\frac{f}{q}\{h\sin L - k\cos L\} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\cos L}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{s^2\sin L}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}}\frac{1}{q}\{h\sin L - k\cos L\} \end{bmatrix} \tag{5.32}$$

and the vector

$$\mathbf{b}^{\mathsf{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & \sqrt{\mu p}\left(\frac{g}{p}\right)^2 \end{bmatrix}, \tag{5.33}$$

where

$$q = 1 + f\cos L + g\sin L, \tag{5.34}$$

$$r = \frac{p}{q}, \tag{5.35}$$

$$\alpha^2 = h^2 - k^2, \tag{5.36}$$

$$\chi = \sqrt{h^2 + k^2}, \tag{5.37}$$

$$s^2 = 1 + \chi^2. \tag{5.38}$$

The equinoctial coordinates $\mathbf{y}$ are related to the Cartesian state $(\mathbf{r}, \mathbf{v})$ according to the expressions

$$\mathbf{r} = \begin{bmatrix} \frac{r}{s^2}\left(\cos L + \alpha^2\cos L + 2hk\sin L\right) \\ \frac{r}{s^2}\left(\sin L - \alpha^2\sin L + 2hk\cos L\right) \\ \frac{2r}{s^2}\left(h\sin L - k\cos L\right) \end{bmatrix}, \tag{5.39}$$

$$\mathbf{v} = \begin{bmatrix} -\frac{1}{s^2}\sqrt{\frac{\mu}{p}}\left(\sin L + \alpha^2\sin L - 2hk\cos L + g - 2fhk + \alpha^2 g\right) \\ -\frac{1}{s^2}\sqrt{\frac{\mu}{p}}\left(-\cos L + \alpha^2\cos L + 2hk\sin L - f + 2ghk + \alpha^2 f\right) \\ \frac{2}{s^2}\sqrt{\frac{\mu}{p}}\left(h\cos L + k\sin L + fh + gk\right) \end{bmatrix}. \tag{5.40}$$

As a result of this transformation, the disturbing acceleration vector $\mathbf{a}_d$ in (5.25) is replaced by

$$\boldsymbol{\Delta} = \boldsymbol{\Delta}_g + \boldsymbol{\Delta}_T \tag{5.41}$$

with a contribution due to oblate earth effects $\boldsymbol{\Delta}_g$ and another caused by thrust $\boldsymbol{\Delta}_T$. The disturbing acceleration is expressed in a rotating radial frame whose principle axes are defined by

$$\mathbf{Q}_r = \begin{bmatrix} \mathbf{i}_r & \mathbf{i}_\theta & \mathbf{i}_h \end{bmatrix} = \begin{bmatrix} \frac{\mathbf{r}}{\|\mathbf{r}\|} & \frac{(\mathbf{r}\times\mathbf{v})\times\mathbf{r}}{\|\mathbf{r}\times\mathbf{v}\|\|\mathbf{r}\|} & \frac{\mathbf{r}\times\mathbf{v}}{\|\mathbf{r}\times\mathbf{v}\|} \end{bmatrix}. \tag{5.42}$$

As stated, (5.28)–(5.31) are perfectly general and describe the motion of a point mass when subject to the disturbing acceleration vector $\boldsymbol{\Delta}$. Notice that when the disturbing acceleration is zero, $\boldsymbol{\Delta} = 0$, the first five equations are simply $\dot{p} = \dot{f} = \dot{g} = \dot{h} = \dot{k} = 0$, which implies that the elements are constant. It is important to note that the disturbing acceleration vector can be attributed to any perturbing force(s). A more complete derivation of the equinoctial dynamics can be found in [11].

5.3.2 Gravitational Disturbing Acceleration

Oblate gravity models are typically defined in a local horizontal reference frame, that is,

$$\boldsymbol{\delta g} = \delta g_n \mathbf{i}_n - \delta g_r \mathbf{i}_r, \tag{5.43}$$

where

$$\mathbf{i}_n = \frac{\mathbf{e}_n - (\mathbf{e}_n^\top \mathbf{i}_r)\mathbf{i}_r}{\|\mathbf{e}_n - (\mathbf{e}_n^\top \mathbf{i}_r)\mathbf{i}_r\|} \tag{5.44}$$

defines the local north direction with $\mathbf{e}_n = (0, 0, 1)$. A reasonably accurate model is obtained if the tesseral harmonics are ignored and only the first four zonal harmonics are included in the geopotential function. In this case, the oblate earth perturbations to the gravitational acceleration are given by

$$\delta g_n = -\frac{\mu \cos\phi}{r^2} \sum_{k=2}^{4} \left(\frac{R_e}{r}\right)^k P_k' J_k, \tag{5.45}$$

$$\delta g_r = -\frac{\mu}{r^2} \sum_{k=2}^{4} (k+1) \left(\frac{R_e}{r}\right)^k P_k J_k, \tag{5.46}$$

where ϕ is the geocentric latitude, R_e is the equatorial radius of the earth, $P_k(\sin\phi)$ is the kth-order Legendre polynomial with corresponding derivative P_k', and J_k are the zonal harmonic coefficients. Finally, to obtain the gravitational perturbations in the rotating radial frame, it follows that

$$\boldsymbol{\Delta}_g = \mathbf{Q}_r^\top \boldsymbol{\delta g}. \tag{5.47}$$

5.3.3 Thrust Acceleration—Burn Arcs

To this point, the discussion has concentrated on incorporating perturbing forces due to oblate earth effects. Of course, the second major perturbation is the thrust acceleration defined by

$$\boldsymbol{\Delta}_T = \frac{g_o T\,[1 + .01\tau]}{w}\mathbf{u}, \tag{5.48}$$

where T is the maximum thrust and $\tau_L \leq \tau \leq 0$ is a throttle factor. In general, the direction of the thrust acceleration vector, which is defined by the time-varying control vector $\mathbf{u}(t) = (u_r, u_\theta, u_h)$, can be chosen arbitrarily as long as the vector has unit length at all points in time. This is achieved using the path constraint (5.30). The magnitude of the thrust is, of course, related to the vehicle weight according to (5.29), where g_0 is the mass to weight conversion factor and the specific impulse of the motor is denoted by I_{sp}. Defining the thrust direction using the vector $\mathbf{u}(t)$ and path constraint $\|\mathbf{u}(t)\| = 1$ is particularly well suited for missions that involve steering over large portions of the trajectory, as illustrated in [9], because ambiguities in the pointing direction are avoided. Specifying the thrust direction by two angles (e.g., yaw and pitch), which are treated as control variables, is not unique since the angles $\alpha = \alpha_0 \pm 2k\pi$ all yield the *same* direction. In contrast, there is a *unique* set of control variables $\mathbf{u}$ corresponding to any thrust direction. This ambiguity is demonstrated in Figure 5.17.

5.3.4 Boundary Conditions

The standard approach for defining the boundary conditions of an orbit transfer problem is to specify the final state in terms of the instantaneous or *osculating* orbit elements at the burnout time t_F. The final orbit for this example has an apogee altitude of 21450 nm, a perigee altitude of 350 nm, an inclination of 63.4 deg, and an argument of perigee of 270 deg. This final orbit can be defined by the boundary conditions (all evaluated at $t = t_F$)

$$p = 40007346.015232 \text{ (ft)}, \tag{5.49}$$

$$\sqrt{f^2 + g^2} = 0.73550320568829, \tag{5.50}$$

$$\sqrt{h^2 + k^2} = 0.61761258786099, \tag{5.51}$$

$$fh + gk = 0, \tag{5.52}$$

$$gh - kf \leq 0. \tag{5.53}$$

The initial orbit for this example is a "standard" space shuttle park orbit, which is circular at an altitude of 150 nm, and an inclination of 28.5 deg. This particular orbit leads to the following values for the equinoctial elements at $t = 0$:

$$p = 21837080.052835 \text{ (ft)}, \qquad f = 0, \qquad g = 0,$$
$$h = -0.25396764647494, \qquad k = 0, \qquad L = \pi \text{ (rad)}.$$

The following constants complete the definition of the problem:

$$w(0) = 1 \text{ (lb)}, \qquad g_0 = 32.174 \text{ (ft/sec}^2\text{)},$$
$$I_{sp} = 450 \text{ (sec)}, \qquad T = 4.446618 \times 10^{-3} \text{ (lb)},$$
$$\mu = 1.407645794 \times 10^{16} \text{ (ft}^3\text{/sec}^2\text{)}, \qquad R_e = 20925662.73 \text{ (ft)},$$
$$J_2 = 1082.639 \times 10^{-6}, \qquad J_3 = -2.565 \times 10^{-6},$$
$$J_4 = -1.608 \times 10^{-6}, \qquad \tau_L = -50.$$

For convenience, we have chosen the initial weight as 1 lb, and the goal is to maximize the final weight, i.e., $w(t_F)$.

5.3.5 Numerical Solution

The solution to this low-thrust orbit transfer problem obtained using the implementation in SOCS is summarized in Figures 5.13 and 5.14. All times are plotted in hours, with the semiparameter p given in millions of feet and the true longitude L in multiples of 360 deg. The iteration history is summarized in Table 5.5. Figure 5.15 illustrates the behavior of the mesh-refinement algorithm. The optimal value for the final weight is $w^*(t_F) = 0.2201791266$ lb, which occurs at $t_F^* = 86810.0$ sec with an optimal throttle parameter value of $\tau^* = -9.09081$. The final trajectory profile is illustrated in Figure 5.16. It is also worthwhile to examine the behavior of the optimal control history when angles are used instead of the direction vector $\mathbf{u}(t)$. In particular, one can define

$$u_r = -\sin\theta,$$
$$u_\theta = \cos\theta\cos\psi,$$
$$u_h = -\cos\theta\sin\psi$$

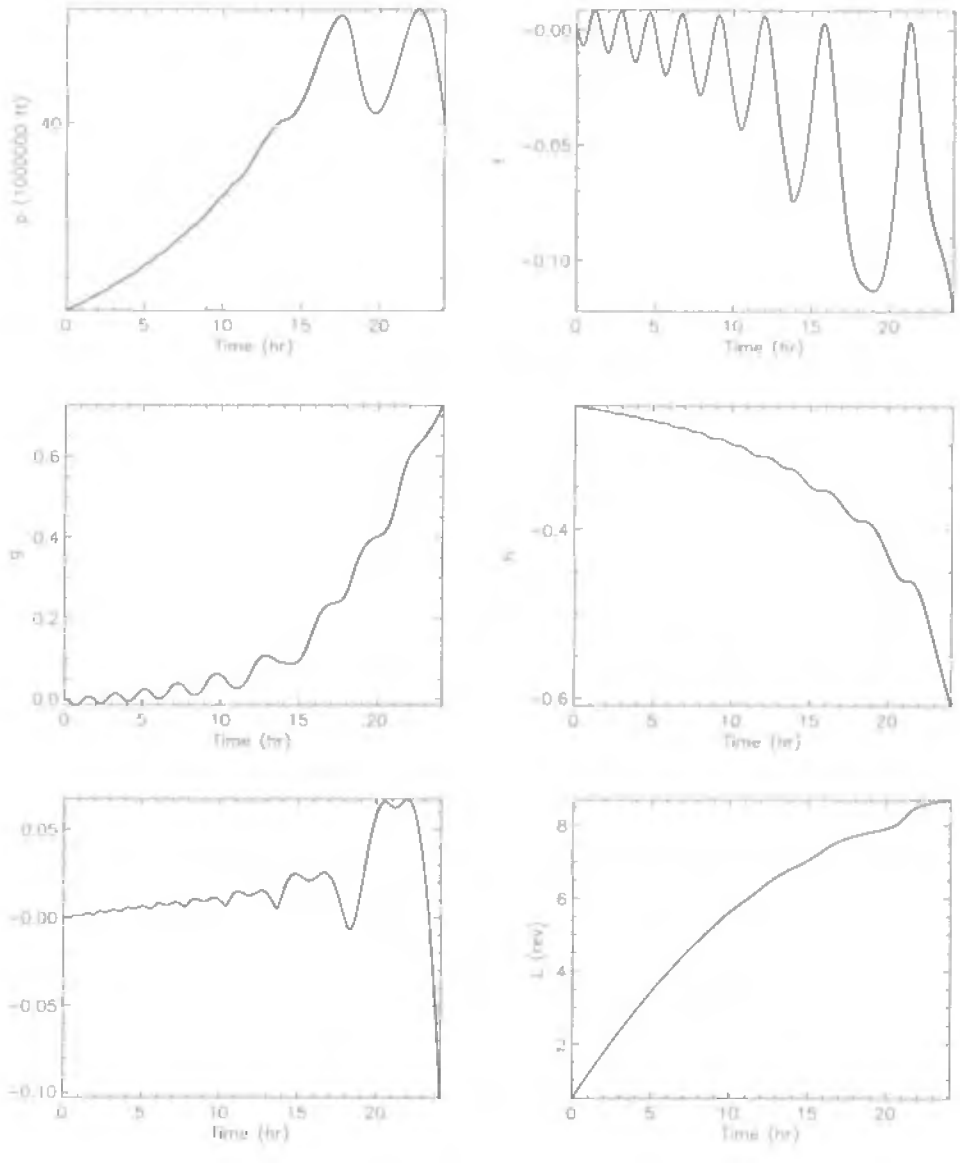


Figure 5.13: Low-thrust transfer—state variables.

using the two control angles (θ, ψ). The time history of these angles illustrated in Figure 5.17 clearly demonstrates the ambiguity in the yaw angle. In fact, if the controls were modeled using angles, it is clear that the mesh-refinement procedure would detect an apparent discontinuity caused by the yaw angle "wrapping" unless the dotted time history was followed.

5.4 Two-Burn Orbit Transfer

In this section, consider a problem similar to Example 5.4, with one important difference—the magnitude of the thrust. In particular, for this example the thrust $T = 1.25$ lb and

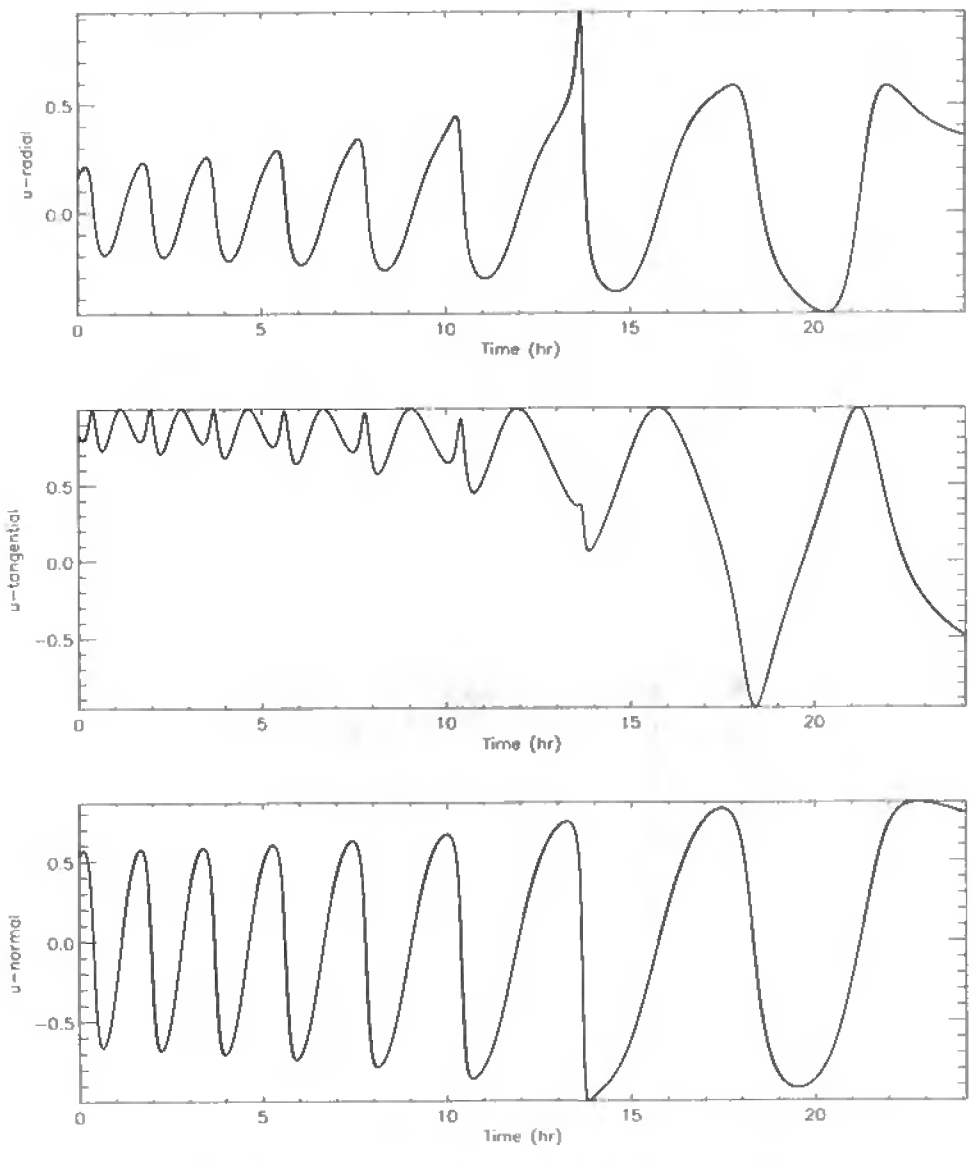


Figure 5.14: Low-thrust transfer—control variables.

Iter.	Disc.	M	GE	HE	FE	RHS	ϵ_{max}	CPU (sec)
1	TR	150	416	80	11231	1684650	$.22 \times 10^{-2}$	0.23×10^{4}
2	TR	299	10	7	604	180596	$.26 \times 10^{-3}$	0.92×10^{2}
3	HS	299	11	8	682	407154	$.31 \times 10^{-4}$	0.31×10^{3}
4	HS	597	8	6	503	600079	$.15 \times 10^{-5}$	0.68×10^{3}
5	HS	966	6	3	292	563852	$.61 \times 10^{-7}$	0.12×10^{4}
Total	-	-	451	104	13312	3436331	-	4562.20

Table 5.5: Low-thrust transfer example.

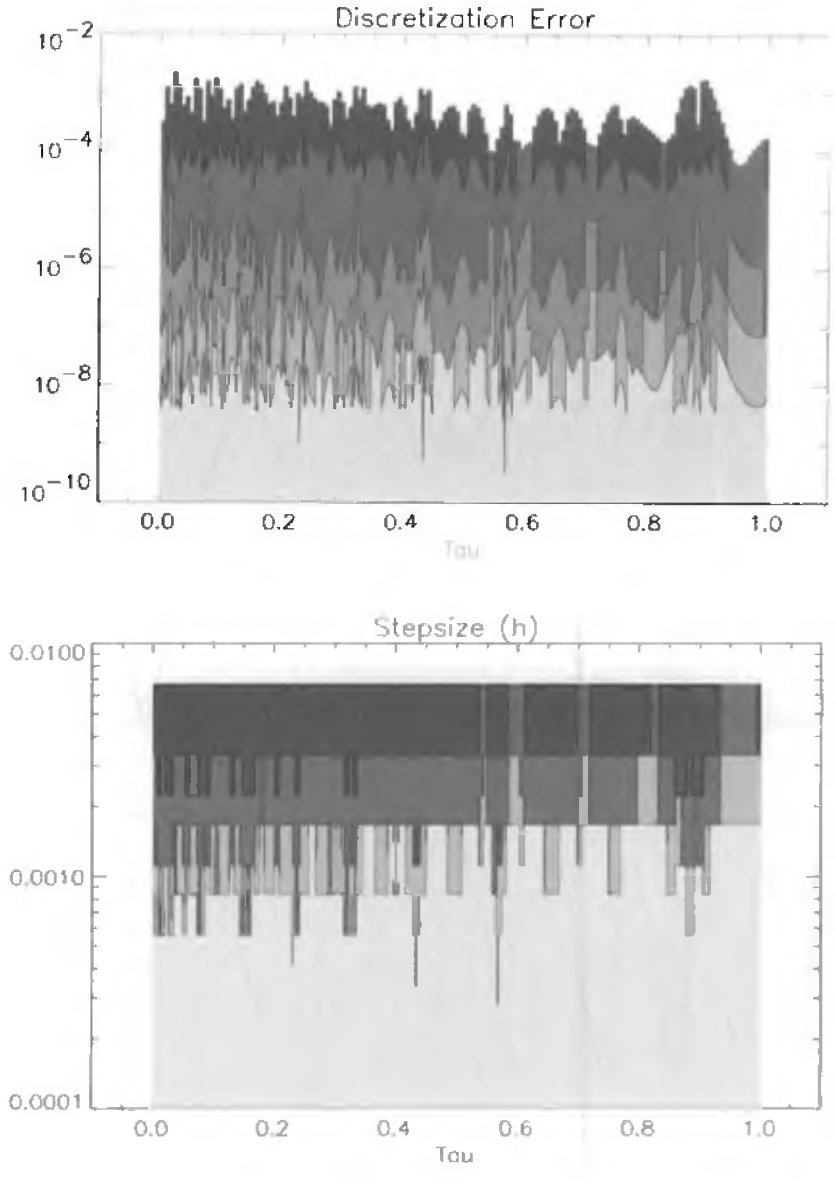


Figure 5.15: Low-thrust transfer— mesh refinement.

the initial weight $w(0) = 1$ lb leading to a description that is very representative of most operational launch vehicles. To make the problem more concrete, let us suppose that the vehicle begins in the same standard space shuttle park orbit as in Example 5.4, which is a 150 nm circular orbit with an inclination of 28.5 deg. The final orbit for this example is chosen to be a circular orbit with an altitude of 19323 nm and an inclination of 0 deg. This orbit is referred to as *geosynchronous* because it has a period of 24 hr, and the motion of a vehicle in this orbit is synchronized with the revolution of the earth beneath it. Thus, a satellite placed in geosynchronous orbit appears motionless when viewed from the earth, making it an attractive platform for communication systems. Since most

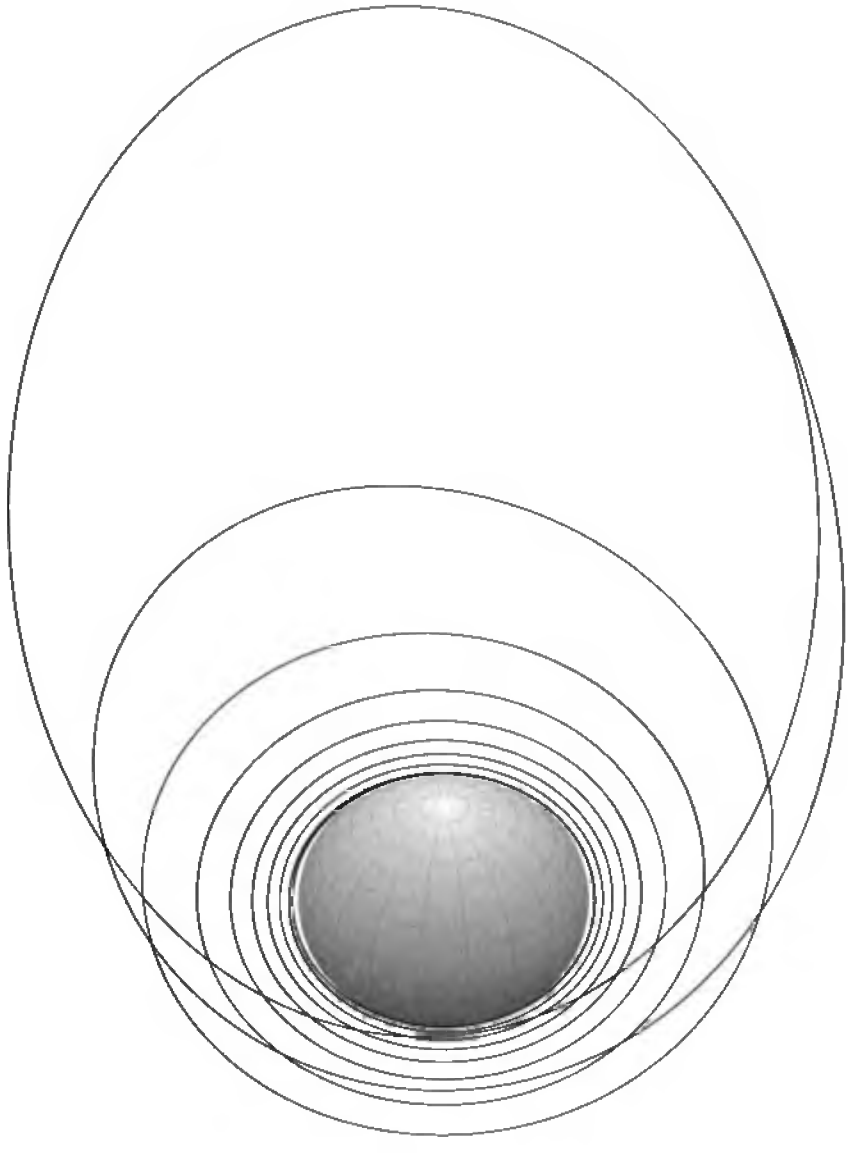


Figure 5.16: Optimal low-thrust transfer.

communication satellites are placed in an orbit of this type, this particular trajectory design problem has been studied extensively, and, in contrast to Example 5.4, which is a hard problem, this may be considered an easy problem. On the other hand, because the problem is rather simple to solve, it permits us to illustrate and compare different solution methods.

The vehicle dynamics are initialized at a point in the park orbit. For convenience, the point at which the vehicle crosses the equator in a northbound direction (referred to as the ascending node) is chosen to be the initial time. After coasting for an unspecified time, the vehicle's engine is ignited. The orientation and duration of this "first burn"

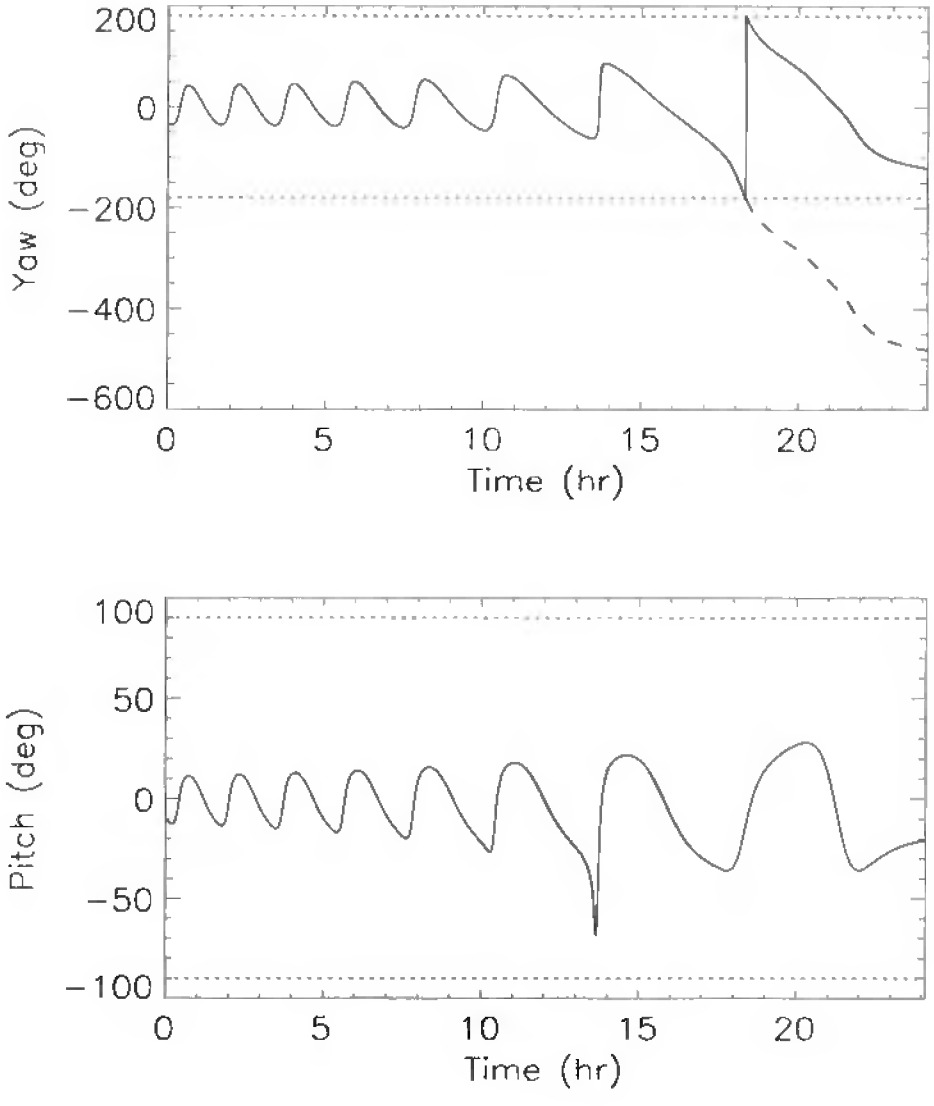


Figure 5.17: Optimal control angles.

are also unspecified. The additional velocity added by the first burn places the vehicle into a "transfer orbit." After coasting for an unspecified time in the transfer orbit, the motor is again ignited and the "second burn" is performed. The duration of the burn and orientation of the vehicle during this time are also unspecified. However, when the second burn is completed, the vehicle must be deployed in the desired geosynchronous orbit. There are a number of ways to quantify an optimal orbit transfer. Typically, the trajectory that minimizes the fuel consumed or maximizes the final weight is preferred. An equivalent approach is to design a trajectory that minimizes the energy added by the propulsion system, and this approach is referred to as a "minimum Δv" transfer.

The motion of a vehicle can be described by the following system of first-order ODEs:

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{5.54}$$
$$\dot{\mathbf{v}} = \mathbf{g} + \mathbf{T}, \tag{5.55}$$
$$\dot{w} = -T/I_{sp}. \tag{5.56}$$

These equations are equivalent to (5.25), where $\mathbf{r}$ is the inertial position vector, $\mathbf{v}$ is the inertial velocity vector, and w is the weight. The gravitational acceleration is defined by $\mathbf{g}$ and the thrust acceleration is defined by $\mathbf{T}$. We use T to denote the magnitude of the thrust $\|\mathbf{T}\|$ and I_{sp} to denote the specific impulse.

5.4.1 Simple Shooting Formulation

Example 5.5. In Section 3.3, we described the *shooting method*, which is often used for solving simple BVPs such as this two-burn transfer. Early attempts to solve this problem introduced approximations to the *physics* in order to expedite the solution of (5.54) (5.56). Although a simple shooting formulation can be obtained without approximating the physics. we will follow the historical technique.

When the thrust T is large, the duration of the burns is very short compared to the overall transfer time. Thus, it is common to assume the burns occur instantaneously. Using this approximation, the net effect of a burn is to change the *velocity* but not the *position*. This technique is called an *impulsive* Δv approximation. To be more precise,

$$\mathbf{v}(t_2) = \mathbf{v}(t_1) + \mathbf{\Delta v}, \tag{5.57}$$

where $\mathbf{v}(t_1)$ is the velocity before the burn, $\mathbf{v}(t_2)$ is the velocity after the burn, $\mathbf{\Delta v}$ is the velocity added by the burn, and $t_1 = t_2$. The velocity change is related to the weight by the expression

$$\|\mathbf{\Delta v}\| = g_0 I_{sp} \ln\left[\frac{w(t_1)}{w(t_2)}\right], \tag{5.58}$$

where $g_0 = 32.174$ (ft/sec^2) is the mass to weight conversion factor. Also, for convenience, we can define the impulsive velocity using spherical coordinates $(\Delta v, \theta, \psi)$, where

$$\mathbf{\Delta v} = \mathbf{Q}_v \begin{pmatrix} \Delta v \cos\theta \cos\phi \\ \Delta v \cos\theta \sin\phi \\ \Delta v \sin\theta \end{pmatrix} \tag{5.59}$$

and the orthogonal matrix

$$\mathbf{Q}_v = \left[\frac{\mathbf{v}}{\|\mathbf{v}\|} \quad \frac{\mathbf{v}\times\mathbf{r}}{\|\mathbf{v}\times\mathbf{r}\|} \quad \frac{\mathbf{v}}{\|\mathbf{v}\|} \times \left(\frac{\mathbf{v}\times\mathbf{r}}{\|\mathbf{v}\times\mathbf{r}\|}\right) \right] \tag{5.60}$$

defines the principle axes of an inertial velocity coordinate frame.

During coast portions of the trajectory, $\mathbf{T} = \mathbf{0}$ and the equations of motion (5.54)–(5.56) are just

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{5.61}$$
$$\dot{\mathbf{v}} = \mathbf{g}. \tag{5.62}$$

The only nonlinear quantity in these equations is the gravitational acceleration $\mathbf{g}(\mathbf{r})$. If one assumes that the earth is spherical, then $\mathbf{g}(\mathbf{r}) = -\mu\mathbf{r}/r^3$ and the differential

equations (5.61)–(5.62) have an analytic solution! Thus, given the state at some time t_0, one can analytically propagate to another time t_1, i.e.,

$$\begin{bmatrix} \mathbf{r}(t_0) \\ \mathbf{v}(t_0) \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{r}(t_1) \\ \mathbf{v}(t_1) \end{bmatrix}. \tag{5.63}$$

In effect, this propagation defines a nonlinear mapping between the states at t_0 and t_1, say,

$$\begin{bmatrix} \mathbf{r}(t_1) \\ \mathbf{v}(t_1) \end{bmatrix} = \mathbf{P}\left[\mathbf{r}(t_0), \mathbf{v}(t_0), t_1, t_0\right]. \tag{5.64}$$

Kepler provided the original solution of this propagation problem over 100 years ago, and a complete description of the computational procedure can be found in [51]. More recently, Huffman [76] has extended the analytic propagation to include the most significant oblate earth perturbation (i.e., including the contribution of J_2 in (5.45) and (5.46)).

At this point, it is worthwhile to expand on a subtle but important detail regarding the meaning of analytic propagation. In order to propagate from one *time* to another, it is necessary to solve Kepler's equation. In its simplest form, one must compute E such that

$$n(t - t_0) = E - e \sin E, \tag{5.65}$$

where n and e are constants for the orbit (the mean motion and eccentricity) and t is the propagation time. Typically, this transcendental equation is solved by a Newton iteration. The variable E, called the eccentric anomaly, determines the angular position of the vehicle at time t on the orbit. In fact, for orbital motion, there is an angular change α that will satisfy the equation

$$k\left[\alpha, \Delta t\right] = 0 \tag{5.66}$$

for a specified value of Δt, but it cannot be written explicitly in the form

$$\alpha \sim k^{-1}\left[\Delta t\right].$$

Thus, if we choose to propagate the orbit by specifying the time change, we must solve a transcendental equation. But an "internal" iteration is undesirable, as discussed in Section 1.14. On the other hand, if we choose to propagate by specifying an angular change, no iteration is required! Thus, the proper formulation is to introduce the angular change α as an NLP variable (in addition to t) and then impose the Kepler equation (5.66).

The fundamental idea of combining impulsive Δv and analytic orbit propagation was originally proposed by a German engineer, Walter Hohmann, who published the idea (in German) in Munich, in 1925. Although his original paper described transfers between circular orbits with the same inclination, the term "Hohmann transfer" is now loosely applied to describe any transfer between arbitrary orbits using impulsive approximations.

The boundary conditions that define the desired geosynchronous orbit are

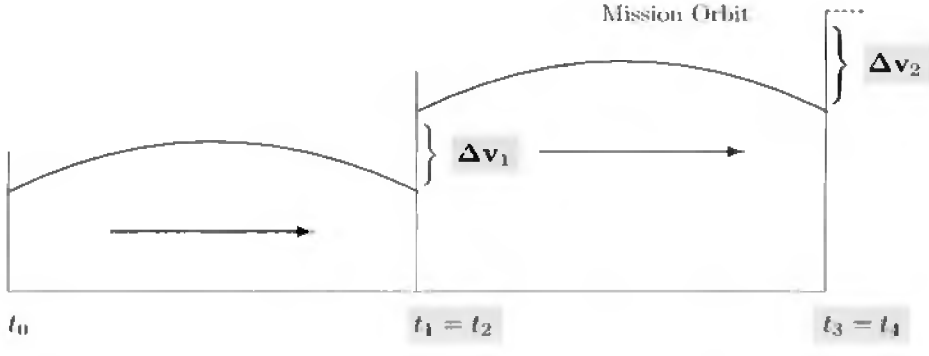
$$\begin{aligned}
(\|\mathbf{r}\| - R_e)/\sigma &= 19323 \text{ (nm)}, && (5.67)\\
\|\mathbf{v}\| &= 10087.5 \text{ (ft/sec)}, && (5.68)\\
\frac{\mathbf{v}^T\mathbf{r}}{\|\mathbf{v}\|\|\mathbf{r}\|} &= 0, && (5.69)\\
\frac{v_3}{\|\mathbf{v}\|} &= 0, && (5.70)\\
\frac{r_3}{\|\mathbf{r}\|} &= 0, && (5.71)
\end{aligned}$$

where the constant $\sigma = 6076.1154855643$ ft/nm. We use the same values for R_e, μ, and J_2 as in Example 5.4.

We can now define the NLP problem that must be solved. The NLP variables are

$$\mathbf{x} = \begin{bmatrix} t_1 \\ \alpha_1 \\ \Delta v_1 \\ \theta_1 \\ \psi_1 \\ t_3 \\ \alpha_2 \\ \Delta v_2 \\ \theta_2 \\ \psi_2 \end{bmatrix} = \begin{bmatrix} \text{First-burn ignition time} \\ \text{First coast angle} \\ \text{First-burn velocity} \\ \text{First-burn pitch angle} \\ \text{First-burn yaw angle} \\ \text{Second-burn ignition time} \\ \text{Second coast angle} \\ \text{Second-burn velocity} \\ \text{Second-burn pitch angle} \\ \text{Second-burn yaw angle} \end{bmatrix} \tag{5.72}$$

When values are specified for the NLP variables, it is possible to compute the objective and constraint functions. This is illustrated schematically below:



For this direct shooting formulation, the function generator is as follows:

Direct Shooting

Input: $\mathbf{x}$
Initialize State:
$\mathbf{r}(t_0), \mathbf{v}(t_0), t_0$
First Coast:
Propagate through angle α_1; from (5.64) compute $\mathbf{r}(t_1), \mathbf{v}(t_1)$.
Constraint Evaluation:
Compute Kepler constraint from (5.66) using α_1, t_1.
First Burn: Given $\Delta v_1, \theta_1, \psi_1$, set $\mathbf{r}(t_2) = \mathbf{r}(t_1)$, $t_2 = t_1$, and compute $\mathbf{v}(t_2)$ from (5.57), (5.59), and (5.60).
Second Coast:
Propagate through angle α_2; from (5.64) compute $\mathbf{r}(t_3), \mathbf{v}(t_3), t_3$.
Constraint Evaluation:
Compute Kepler constraint from (5.66) using α_2, t_3.
Second Burn: Given $\Delta v_2, \theta_2, \psi_2$, set $\mathbf{r}(t_4) = \mathbf{r}(t_3)$, $t_4 = t_3$, and compute $\mathbf{v}(t_4)$ from (5.57), (5.59), and (5.60).
Constraint Evaluation:
Compute boundary conditions at t_4 from (5.67)–(5.71).
Terminate Trajectory
Compute objective $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$.
Output: $F(\mathbf{x}), \mathbf{c}(\mathbf{x})$

The goal is to minimize the objective function

$$F(\mathbf{x}) = \Delta v_1 + \Delta v_2. \tag{5.73}$$

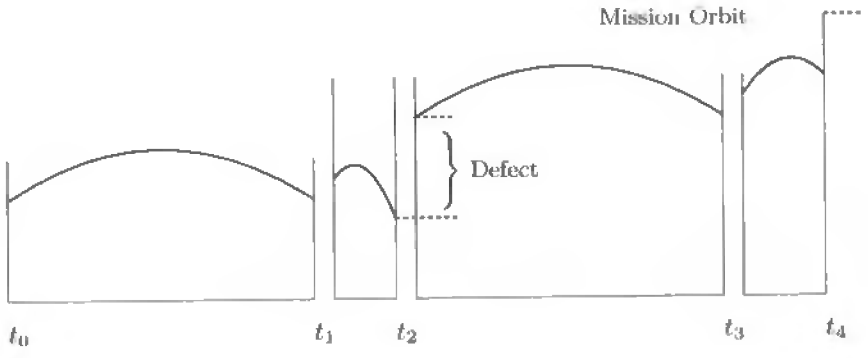
Since none of the computed quantities for this example explicitly depends on time, this problem can also be formulated using only propagation angles. Table 5.6 presents the solutions obtained with and without time as a propagation variable (formulations A and B, respectively). The only apparent difference between these approaches is that the formulation B requires fewer function evaluations (FE), which is not surprising because there are fewer variables. The equivalent weight delivered to the final orbit can be computed using (5.58) and is $w(t_F) = 0.23680470963252$ lb.

	Formulation A	Formulation B
t_1	2690.41	*
α_1	179.667	179.667
Δv_1	8049.57	8049.57
θ_1	0.148637×10^{-2}	0.146647×10^{-2}
ψ_1	−9.08446	−9.08445
t_3	21658.5	*
α_2	180.063	180.063
Δv_2	5854.61	5854.61
θ_2	-0.136658×10^{-2}	-0.135560×10^{-2}
ψ_2	49.7892	49.7892
$\Delta v_1 + \Delta v_2$	13904.18221	13904.18220
FE	294	249

Table 5.6: Minimum Δv transfer.

5.4.2 Multiple Shooting Formulation

Example 5.6. A very natural partition of the problem is suggested by the physics of this two-burn orbit transfer. Modeling the dynamics using four distinct phases, namely the coast in the park orbit, the first burn, the coast in the transfer orbit, and the second burn, is an obvious choice. We can also treat each of these phases as segments and apply the *multiple shooting* method as described in Section 3.4. This is illustrated below:



For the multiple shooting formulation, one must augment the NLP variables (5.72) to include the state on each side of the segment (phase) boundaries as illustrated. Thus, we include as new NLP variables the states $\mathbf{r}(t_k^-), \mathbf{v}(t_k^-)$ and $\mathbf{r}(t_k^+), \mathbf{v}(t_k^+)$ for $k = 1, 2, 3, 4$, where $-$ denotes the quantity before the boundary and $+$ denotes the quantity after the boundary. Additional "defect" constraints must be imposed to ensure continuity across the segment boundaries and also to guarantee that the analytically propagated state is consistent with the (new) guessed values. Thus, when compared to the direct shooting method, the number of variables increases from 10 to 63. However, the number of constraints also increases from 7 to 60, so that the total number of degrees of freedom is unchanged.

One of the primary reasons to use a multiple shooting method is to improve robustness, which is demonstrated even on this simple four-segment example. As shown in Table 5.7, the multiple shooting method solved the problem with fewer gradient evaluations (2), fewer Hessian evaluations (3), and fewer function evaluations (97) when compared to the simple shooting approach. Even the modest increase in CPU time (attributed to increased problem size) is somewhat deceiving because the cost of evaluating the functions is so small for this example.

	Shooting	Multiple shooting	Δ
GE	11	9	−22%
HE	5	2	−150%
FE	294	197	−49%
CPU (sec)	0.699127	0.872283	+20%

Table 5.7: Shooting versus multiple shooting.

5.4.3 Collocation Formulation

Example 5.7. Both Example 5.5 and Example 5.6 used approximate *physics*, that is, simplified orbit dynamics and impulsive burn approximations. For comparison, let us now solve the same orbit transfer without making these simplifying assumptions. We still pose the problem using four phases. However, in this example we consider finite-duration burn phases with optimal steering. Oblate gravitational accelerations will be used throughout, and we will describe the dynamics using the previously discussed equinoctial coordinates. During the coast phases, from (5.28), the dynamics are given by

$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta}_g + \mathbf{b}, \tag{5.74}$$

where $\mathbf{A}(\mathbf{y})$ is defined by (5.32), $\mathbf{b}$ by (5.33), and $\boldsymbol{\Delta}_g$ by (5.47). During the burn phases,

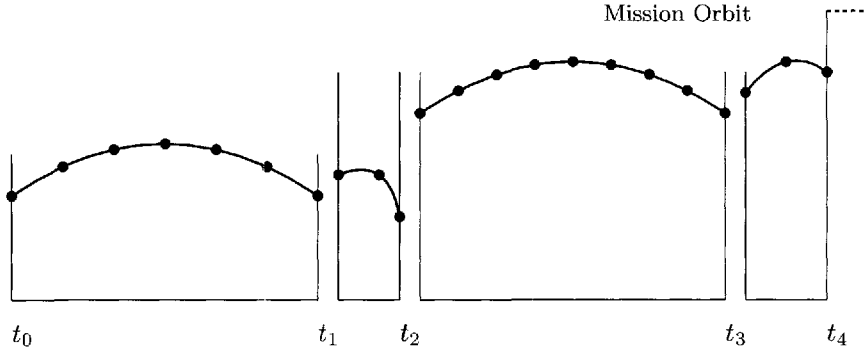
$$\dot{\mathbf{y}} = \mathbf{A}(\mathbf{y})\boldsymbol{\Delta} + \mathbf{b}, \tag{5.75}$$

$$\dot{w} = -T/I_{sp}, \tag{5.76}$$

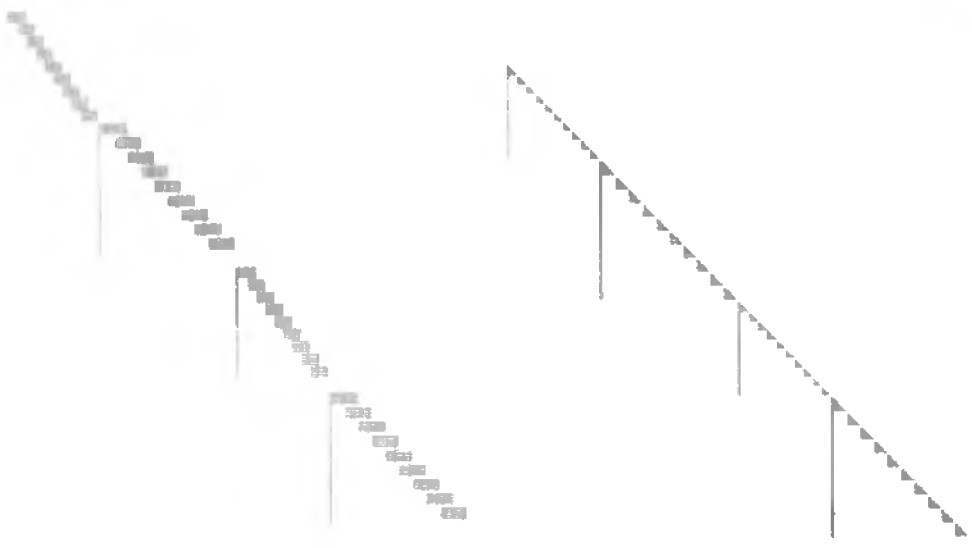
and the total perturbation is given by (5.41). To be consistent with Examples 5.5 and 5.6, we define the orientation of the thrust using pitch and yaw angles in the inertial velocity frame and then transform them to the radial frame, that is,

$$\boldsymbol{\Delta}_T = \mathbf{Q}_r^{\mathsf{T}}\mathbf{Q}_v \begin{bmatrix} T\cos\theta\cos\phi \\ T\cos\theta\sin\phi \\ T\sin\theta \end{bmatrix}, \tag{5.77}$$

where $\mathbf{Q}_v$ is defined by (5.60) and $\mathbf{Q}_r$ is given by (5.42). Since the dynamics are represented using equinoctial coordinates, it is convenient to also specify the boundary conditions in these coordinates. Thus, the geosynchronous conditions (5.67)–(5.71) are equivalent to having $p = 19323/\sigma + R_e$ and $f = g = h = k = 0$. Constraints are also used to link the equinoctial states across the phase boundaries as with the multiple shooting formulation. The weight at the end of phase 2 is also constrained to equal the weight at the beginning of phase 4. The situation is illustrated below:



SOCS was used to solve the problem beginning with a trapezoidal discretization and 10 grid points per phase. The first NLP has 307 variables and 268 active constraints, or 39 degrees of freedom. Figure 5.18 displays the sparsity pattern of the Jacobian and Hessian matrices, corresponding to the initial trapezoidal discretization with 10 grid points per phase. Note that, because the Hessian is symmetric, only the lower-triangular portion is shown. The iteration history is summarized in Table 5.8. The second and third columns in the table give a phase-by-phase breakdown of the discretization



(a) Jacobian (b) Hessian

Figure 5.18: Sparsity patterns.

Iter.	Disc.	M	FE	$\epsilon_{\max}$	T (sec)
1	(T,T,T,T)	(10,10,10,10)	2164	0.26×10^{0}	0.41×10^{2}
2	(H,T,T,T)	(10,19,16,19)	604	0.53×10^{-2}	0.20×10^{2}
3	(H,H,H,H)	(19,19,16,19)	526	0.14×10^{-2}	0.32×10^{2}
4	(H,H,H,H)	(37,37,31,37)	137	0.96×10^{-5}	0.24×10^{2}
5	(H,H,H,H)	(73,73,61,37)	113	0.36×10^{-6}	0.35×10^{2}
6	(H,H,H,H)	(145,73,121,37)	113	0.59×10^{-7}	0.49×10^{2}
Total	-	376	3657		201.84

Table 5.8: Two-burn orbit transfer performance summary.

method (Disc.) and number of grid points (M), respectively. Trapezoidal discretization is denoted by T and separated Simpson by H. Observe that the mesh-refinement algorithm tends to concentrate a large number of points in the early portions of the trajectory because the oblate earth perturbations are more significant in this region. The final mesh-refinement iteration required solving a sparse NLP problem with 5149 variables, 4714 active constraints, and 435 degrees of freedom.

Figure 5.19 illustrates the optimal two-burn transfer to geosynchronous orbit. The optimal steering angles for both burns are plotted in Figure 5.20 with the times normalized to the beginning of the burn. It is interesting to observe that the optimal burn times, i.e., the lengths of phases 2 and 4, are $t_2 - t_1 = 141.47$ (sec) and $t_4 - t_3 = 49.40$ (sec). When compared to the total mission time of 21683.5 (sec), these burn times are quite short, hence justifying the impulsive approximation. It is also interesting to note that the "true" optimal objective $w(t_F) = 0.2367248713$ lb is only 0.033% less than the impulsive burn analytic coast approximation computed in Example 5.5. However, it is necessary to use the optimal steering in Figure 5.20 to achieve this performance.

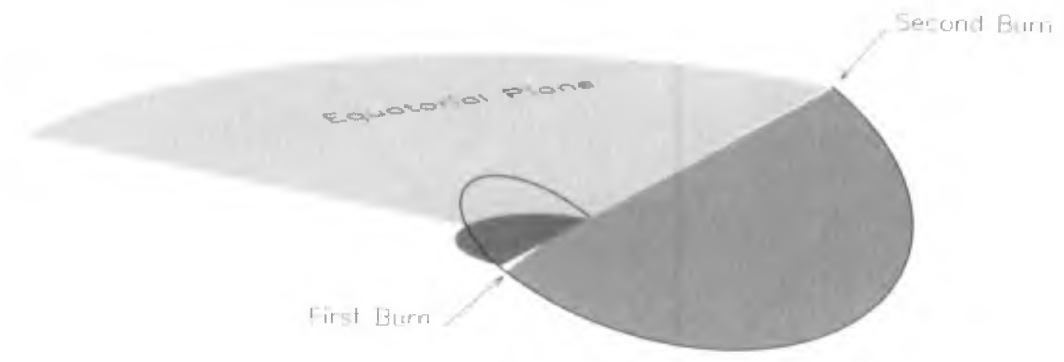
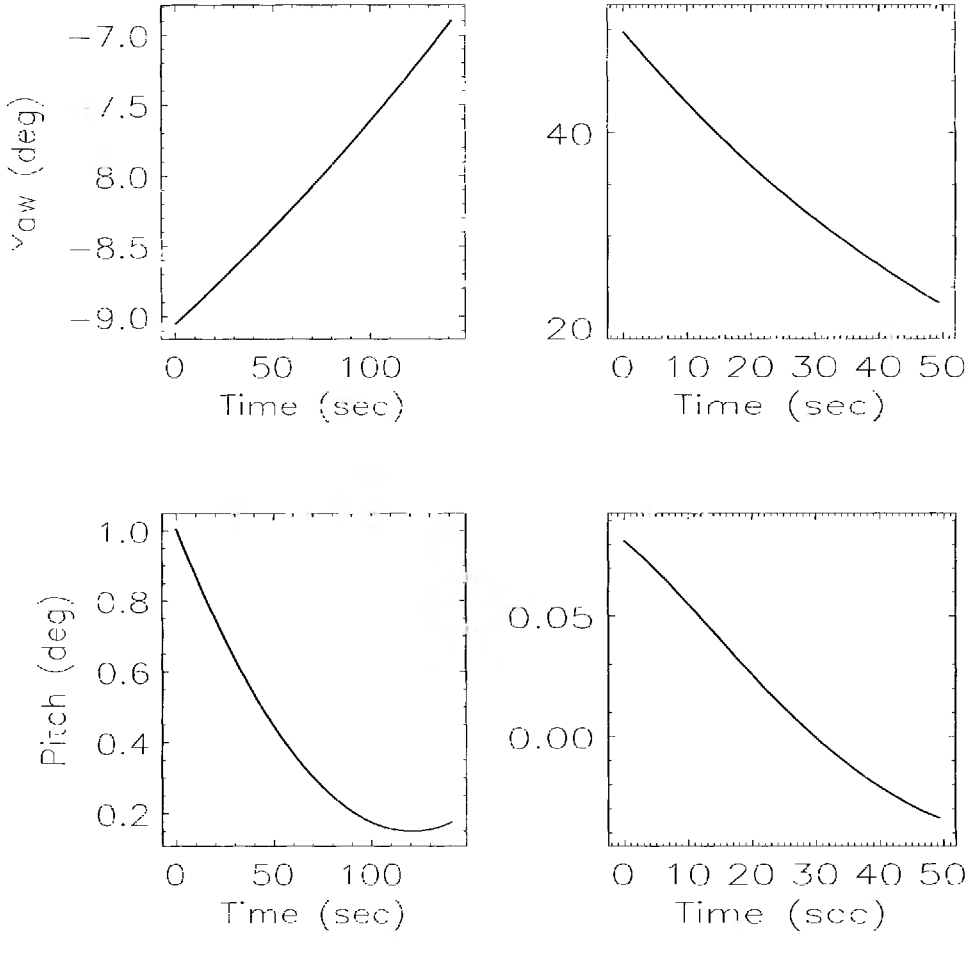


Figure 5.19: Two-burn orbit transfer.



(a) First burn

(b) Second burn

Figure 5.20: Optimal control angles.

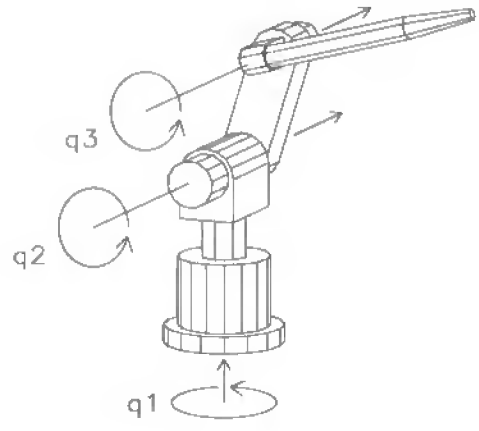


Figure 5.21: Manutec r3 robot.

5.5 Industrial Robot

Example 5.8. An interesting example describing the motion of an industrial robot is presented in [105]. The machine is shown in Figure 5.21 and is called the Manutec r3 [86]. It has six degrees of freedom, although only three degrees of freedom are considered in this formulation since they adequately describe the position of the tool center point. The dynamics are described by the second-order system

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{f}(\dot{\mathbf{q}}, \mathbf{q}. \mathbf{u}), \tag{5.78}$$

where the vector $\mathbf{q} = [q_1(t), q_2(t), q_3(t)]^\mathsf{T}$ defines the relative angles between the robot arms. The torque voltages $\mathbf{u} = [u_1(t), u_2(t), u_3(t)]^\mathsf{T}$ are applied to control the robot motors. The symmetric positive definite mass matrix $\mathbf{M}(\mathbf{q})$ involves very complicated expressions of the state $\mathbf{q}$, as does the function $\mathbf{f}$, which defines the moments caused by Coriolis, centrifugal, and gravitational forces. A more complete description of the quantities in this *multibody system* can be found in [105]. If the angular velocities are denoted by $\mathbf{v}$. then (5.78) can be written as the first-order system

$$\dot{\mathbf{q}} = \mathbf{v}, \tag{5.79}$$

$$\mathbf{M}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{f}(\mathbf{v}, \mathbf{q}, \mathbf{u}). \tag{5.80}$$

Because the matrix $\mathbf{M}(\mathbf{q})$ has a special tree structure, it is possible to construct the inverse using an $\mathcal{O}(n)$ algorithm ($n = 3$), as described in [86]. Thus, we obtain the semi-explicit first-order system

$$\dot{\mathbf{q}} = \mathbf{v}, \tag{5.81}$$

$$\dot{\mathbf{v}} = \mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v}, \mathbf{q}, \mathbf{u}). \tag{5.82}$$

It should be emphasized that the right-hand side of (5.82) involves *very complicated* expressions that are often generated automatically by simulation software systems. The computational results use software graciously provided by O. von Stryk to compute these differential equations.

The goal is to construct a trajectory for the robot tool tip that goes from one specified location to another (a "point-to-point" path). Also, we want the tool to begin and end

the trajectory at rest, i.e., with zero velocity. Thus, we have the boundary conditions

$$\begin{aligned} \mathbf{q}(0) &= (0, -1.5, 0)^\mathsf{T} \text{ (rad)}, & \mathbf{v}(0) &= \mathbf{0} \text{ (rad/sec)}, \\ \mathbf{q}(t_F) &= (1, -1.95, 1)^\mathsf{T} \text{ (rad)}, & \mathbf{v}(t_F) &= \mathbf{0} \text{ (rad/sec)}. \end{aligned}$$

This particular tool also has physical limits that restrict the state and control variables. Specifically,

$$\begin{aligned} \|\mathbf{u}(t)\|_\infty &\leq 7.5 \text{ (V)}, \\ |q_1(t)| &\leq 2.97 \text{ (rad)}, \\ |q_2(t)| &\leq 2.01 \text{ (rad)}, \\ |q_3(t)| &\leq 2.86 \text{ (rad)}, \\ |v_1(t)| &\leq 3.0 \text{ (rad/sec)}, \\ |v_2(t)| &\leq 1.5 \text{ (rad/sec)}, \\ |v_3(t)| &\leq 5.2 \text{ (rad/sec)}. \end{aligned}$$

Three different objective functions are considered in [105], the first two being minimum time

$$J = t_F \tag{5.83}$$

and minimum energy

$$J = \int_0^{t_F} \mathbf{u}^\mathsf{T}(t)\mathbf{u}(t)dt \tag{5.84}$$

for a fixed final time t_F.

The minimum energy problem is relatively straightforward and of significant practical value. However, a number of the concepts described in this book are illustrated by the minimum time problem and so it will be the primary focus of the discussion. This example is useful because, as the authors suggest in their abstract,

> The highly accurate solutions reported in this paper may also serve as benchmark problems for other methods.

What is most admirable is *how* the authors obtained the solutions using an indirect multiple shooting method. As stated in [105],

> The main drawbacks when applying the multiple shooting method in the numerical solution of optimal control problems are, 1. the derivation of the necessary conditions (e.g., the adjoint differential equations), 2. the estimation of the optimal switching structure, and 3. the estimation of an appropriate initial guess of the unknown state and adjoint variables $x(t), \lambda(t), \eta(t)$ in order to start the iteration process.

Briefly, their approach was to derive the adjoint equations using the software tool MAPLE, which produced over 4000 lines of FORTRAN code. Then a direct transcription method (with a coarse mesh) was used to construct both an initial guess for the switching structure and estimates for the state and adjoint variables. Finally, this information was used to initialize an indirect multiple shooting method. Our goal here is to

solve the same problem *without* deriving the adjoint equations and guessing the adjoint variables.

As posed, the minimum time problem presents two major difficulties. First, the control variable appears linearly in the differential equations. Consequently, it is most likely that the control will be bang-bang, as discussed in Example 4.7. Singular subarcs, as described in Example 4.5, are also possible. However, for the specific boundary conditions given here, they do not appear. A more complete analysis can be found in [105]. Second, when the limits on the angular velocity are active, the resulting DAE has index two. Since the Jacobian matrix is singular on the state constraints, the control variable is not well defined by the usual necessary conditions, as illustrated by Example 4.6. The approach we will follow is to first estimate the switching structure and then solve a multiphase problem with index reduction on the phases corresponding to the active state boundaries.

The first step is to compute an estimate for the switching structure. To do this, let us solve a modified version of the minimum time problem. Specifically, minimize

$$J = t_F + \rho \int_0^{t_F} \mathbf{u}^{\mathsf{T}}(t)\mathbf{u}(t)dt, \tag{5.85}$$

where the "small" parameter $\rho = 1 \times 10^{-5}$ is chosen to regularize the problem. By introducing this quadratic regularization, the control becomes uniquely defined and the solution should be "close" to the minimum time problem. It is not necessary (or desirable) to solve this modified problem to a high degree of precision because the primary goal is to construct the appropriate switching structure. Table 5.9 summarizes the results, which were intentionally terminated after seven mesh-refinement iterations. In fact, the computed approximation to the minimum time is 0.495196288 sec, which agrees with the true optimum value 0.49518904 sec to four significant figures. Although there is very little "visible" difference in the objective function, there is a visible difference in the control history. It is interesting to note that the mesh-refinement algorithm presented in Section 4.6.9 selected the separated Simpson discretization because the function evaluations were relatively expensive.

Iter.	Disc.	M	GE	HE	FE	RHS	$\epsilon_{\max}$	CPU (sec)
1	TR	10	16	10	619	6190	0.11×10^{-1}	0.27×10^{1}
2	TR	19	10	8	469	8911	0.86×10^{-2}	0.44×10^{1}
3	HS	19	7	5	304	11248	0.23×10^{-2}	0.14×10^{2}
4	HS	37	7	5	304	22192	0.20×10^{-2}	0.37×10^{2}
5	HS	46	8	6	359	32669	0.36×10^{-3}	0.57×10^{2}
6	HS	84	7	5	304	50768	0.50×10^{-4}	0.19×10^{3}
7	HS	97	5	3	194	37442	0.13×10^{-4}	0.19×10^{3}
Total		97	60	42	2553	169420		493.53

Table 5.9: Minimum time with regularization.

The solution to the relaxed problem is illustrated by the dashed line in Figure 5.22. The relative accuracy of the solution is dominated by the discretization error in the neighborhood of the discontinuities. However, instead of trying to improve the accuracy by adding grid points, it is better to explicitly introduce this discontinuous behavior using separate phases. Examination of the solution provides an estimate for the switching

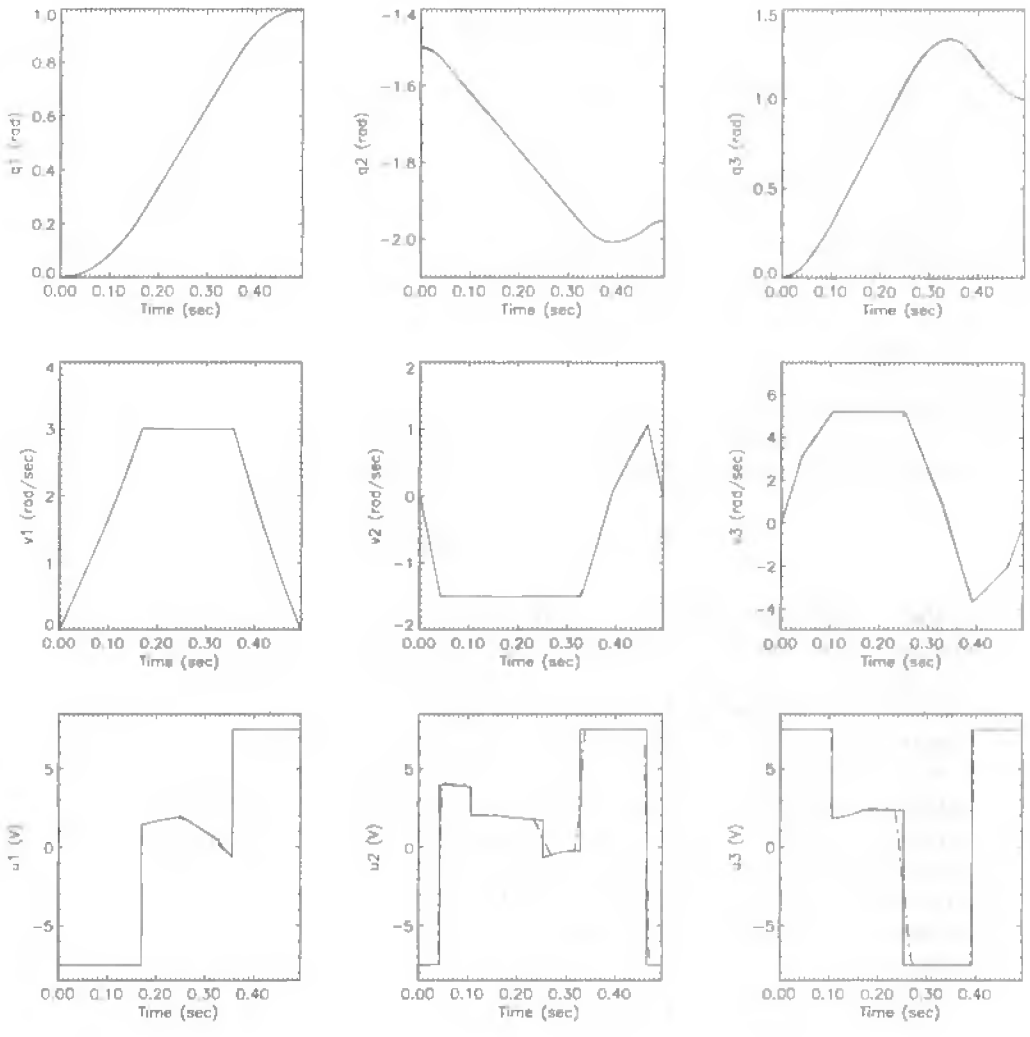


Figure 5.22: Robot solution.

structure. Thus, we are led to model the behavior using nine distinct phases with each phase characterized by a different set of active path constraints. Table 5.10 summarizes the phase-by-phase situation. Thus, in phase 1, the first three angular velocities v_1, v_2, and v_3 are free, whereas the first two control variables u_1 and u_2 are at their minimum values, while the final control u_3 is a maximum.

To illustrate how the multiple-phase structure is exploited, let us consider the dynamics in phase 3. In phase 3, state variable v_2 is on its lower bound and state variable v_3 is on its upper bound. Thus, we must impose the algebraic constraints

$$0 = v_2(t) - v_{2L}, \tag{5.86}$$

$$0 = v_3(t) - v_{3U}, \tag{5.87}$$

where the lower bound $v_{2L} = -1.5$ and the upper bound $v_{3U} = 5.2$. Neither (5.86) nor (5.87) explicitly contains a control variable and, as such, both are index-two path

Phase	v_1	v_2	v_3	u_1	u_2	u_3
1	free	free	free	min	min	max
2	free	min	free	min	free	max
3	free	min	max	min	free	free
4	max	min	max	free	free	free
5	max	min	free	free	free	min
6	max	free	free	free	max	min
7	free	free	free	max	max	min
8	free	free	free	max	max	max
9	free	free	free	max	min	max

Table 5.10: Minimum time switching structure.

constraints. To reduce the index, we must differentiate with respect to time giving

$$0 = \dot{v}_2(t), \tag{5.88}$$
$$0 = \dot{v}_3(t). \tag{5.89}$$

Thus, from (5.81) and (5.82) we must have

$$\dot{q}_1 = v_1(t), \tag{5.90}$$
$$\dot{q}_2 = v_{2L}, \tag{5.91}$$
$$\dot{q}_3 = v_{3U}, \tag{5.92}$$
$$\dot{v}_1 = \left\{\mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v},\mathbf{q},\mathbf{u})\right\}_1, \tag{5.93}$$
$$0 = \left\{\mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v},\mathbf{q},\mathbf{u})\right\}_2, \tag{5.94}$$
$$0 = \left\{\mathbf{M}^{-1}(\mathbf{q})\mathbf{f}(\mathbf{v},\mathbf{q},\mathbf{u})\right\}_3. \tag{5.95}$$

Since $\mathbf{v} = (v_1(t), v_{2L}, v_{3U})$ and $\mathbf{u} = (u_{1L}, u_2(t), u_3(t))$, the dynamics on phase 3 are completely defined by this (index-one) DAE system. Thus, phase 3 can be modeled using four (not six) differential equations, and two (not three) algebraic equations. The differential (state) variables on this phase are

$$(q_1(t), q_2(t), q_3(t), v_1(t))$$

and the algebraic (control) variables are $(u_2(t), u_3(t))$. The beginning and end of the phase are not specified so we must treat the values $t_I^{(3)}$ and $t_F^{(3)}$ as additional NLP variables, where phase 3 is defined on the domain $t_I^{(3)} \le t \le t_F^{(3)}$. The phase description is complete when boundary conditions are specified. Continuity at the beginning and end of the phase is enforced by imposing the boundary conditions

$$t_F^{(2)} = t_I^{(3)}, \qquad t_F^{(3)} = t_I^{(4)},$$
$$q_1(t_F^{(2)}) = q_1(t_I^{(3)}), \qquad q_1(t_F^{(3)}) = q_1(t_I^{(4)}),$$
$$q_2(t_F^{(2)}) = q_2(t_I^{(3)}), \qquad q_2(t_F^{(3)}) = q_2(t_I^{(4)}),$$
$$q_3(t_F^{(2)}) = q_3(t_I^{(3)}), \qquad q_3(t_F^{(3)}) = q_3(t_I^{(4)}),$$
$$v_1(t_F^{(2)}) = v_1(t_I^{(3)}), \qquad v_1(t_F^{(3)}) = v_1(t_I^{(4)}).$$

To ensure the proper transition for the state-constrained variables between phases 2, 3, and 4, at the end of phase 2 we must impose the condition $v_3(t_F^{(2)}) = v_{3U}$ and, at the

Iter.	Disc.	M	GE	HE	FE	RHS	$\epsilon_{\max}$	CPU (sec)
1	TR	45	4	0	34	1530	0.18×10^{-4}	0.34×10^{1}
2	HC	45	2	0	34	2754	0.25×10^{-6}	0.22×10^{1}
3	HC	57	1	0	18	1890	0.66×10^{-7}	0.23×10^{1}
Total		57	7	0	86	6174		7.81

Table 5.11: Minimum time with switching structure.

Problem	Indirect multiple shooting	Direct transcription
Min. energy	20.404247	20.40424581
Min. time	0.49518904	0.495189037

Table 5.12: Accuracy comparison.

end of phase 3, it is necessary that $v_1(t_F^{(3)}) = v_{1U}$. Observe that no continuity conditions are imposed on the control variables. In fact, one benefit of the multiphase treatment of this problem is that it does permit accurate modeling of the control discontinuities.

It is worth noting that the technique presented here models phase 3 with a reduced set of state and control variables. Computationally, this is advantageous because the size of the transcribed problem has been reduced. However, it may be more convenient to implement a problem that has the same number of states and controls on each phase. This can be achieved by adding the required path equality constraints on each phase and then simply solving a larger NLP problem. This method is outlined in the doctoral thesis of O. von Stryk.

For brevity, we omit an explicit description of the approach for all nine phases. However, it should be clear that with a known switching structure it is relatively straightforward to implement the correct conditions in a computational tool such as SOCS. When this information is explicitly incorporated into a nine-phase formulation, the resulting nonlinear program is well-posed and leads to the solution illustrated by the solid line in Figure 5.22. Table 5.11, which summarizes the behavior of the algorithm, clearly demonstrates the benefit of incorporating the switching structure into the formulation.

As a final point of interest, it is worth comparing the accuracy of the solutions computed using the direct transcription approach with those obtained by the benchmark indirect multiple shooting approach [105]. The optimal objective function values are given in Table 5.12 for both methods, with the significant figures that agree underlined. The results agree to seven significant figures in both cases, which also validates the reliability of the discretization error $\epsilon_{\max} \leq 1 \times 10^{-7}$. More important is the fact that the direct transcription results were obtained without computing the adjoint equations and estimating values for the adjoint variables. This is especially encouraging since it suggests that accurate results can be obtained for many practical applications even when it is too complicated to form the adjoint equations.

5.6 Multibody Mechanism

Example 5.9. Hairer and Wanner [68, pp. 530–542] describe a very nice example of a *multibody system* called "Andrew's squeezer mechanism" and have graciously supplied a software implementation of the relevant equations. The problem is used as a benchmark

for testing a number of different multibody simulation codes as described in [96]. For the sake of brevity, we omit a detailed description of the problem and refer the reader to [68]. The multibody dynamics are described by a second-order system similar to (5.78):

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} = \mathbf{f}(\dot{\mathbf{q}}, \mathbf{q}, u) - \mathbf{G}^{\mathsf{T}}(\mathbf{q})\boldsymbol{\lambda}, \tag{5.96}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{q}). \tag{5.97}$$

For this example, the "position" vector $\mathbf{q} = (\beta, \Theta, \gamma, \Phi, \delta, \Omega, \varepsilon)^{\mathsf{T}}$ consists of seven angles that define the orientation of the seven-body mechanism. The bodies are linked together via the algebraic constraints (5.97). The symmetric *mass matrix* $\mathbf{M}$ is tridiagonal, and the derivation of these equations is often facilitated by noting that

$$m_{ij} = \frac{\partial^2 T}{\partial \dot{q}_i \partial \dot{q}_j},$$

where T is the kinetic energy of the system. The mechanism is driven by a motor whose drive torque is given by u. In [68], the torque $u_0 = 0.033$ Nm is treated as a constant, in which case this is simply an IVP. For the sake of illustration, let us assume that the drive torque is a control variable that can be adjusted as a function of time. Let us impose bounds

$$0 \le u(t) \le 0.066 \qquad (= 2 \times 0.033) \tag{5.98}$$

and then try to compute the control such that

$$J = \frac{1}{t_F u_0^2} \int_0^{t_F} u^2(t) dt \tag{5.99}$$

is minimized over the time domain $0 \le t \le t_F = 0.03$ ms.

First, let us convert (5.96) from a second-order implicit form to a first-order semi-explicit DAE system giving

$$\dot{\mathbf{q}} = \mathbf{v}, \tag{5.100}$$

$$\dot{\mathbf{v}} = \mathbf{w}, \tag{5.101}$$

$$\mathbf{0} = \mathbf{M}(\mathbf{q})\mathbf{w} - \mathbf{f}(\mathbf{v}, \mathbf{q}, u) + \mathbf{G}^{\mathsf{T}}(\mathbf{q})\boldsymbol{\lambda}, \tag{5.102}$$

$$\mathbf{0} = \mathbf{g}(\mathbf{q}). \tag{5.103}$$

In this formulation, $\mathbf{w}$, $\boldsymbol{\lambda}$, and u are the algebraic variables $\mathbf{u}$, whereas $\mathbf{q}$ and $\mathbf{v}$ are the differential variables $\mathbf{y}$. Observe that the complete set of algebraic variables denoted by $\mathbf{u}$ includes the "real" control u as well as the other algebraic variables $\mathbf{w}$ and $\boldsymbol{\lambda}$. If we differentiate (5.103) once with respect to time, we find

$$\mathbf{0} = \frac{d}{dt}[\mathbf{g}(\mathbf{q})] = \mathbf{G}(\mathbf{q})\dot{\mathbf{q}} = \mathbf{G}(\mathbf{q})\mathbf{v}. \tag{5.104}$$

A second differentiation with respect to time yields

$$\mathbf{0} = \frac{d^2}{dt^2}[\mathbf{g}(\mathbf{q})] = \frac{d}{dt}[\mathbf{G}(\mathbf{q})\mathbf{v}] = \dot{\mathbf{G}}(\mathbf{q})\mathbf{v} + \mathbf{G}(\mathbf{q})\dot{\mathbf{v}} = \mathbf{g}_{qq}(\mathbf{q})(\mathbf{v}, \mathbf{v}) + \mathbf{G}(\mathbf{q})\mathbf{w}. \tag{5.105}$$

Since the algebraic variable $\mathbf{w}$ appears in this *acceleration-level* constraint, no additional derivatives are needed. Observe that in its original form the problem is index three, and

this illustrates the process of *index reduction*. Collecting results yields the index-one DAE system

$$\dot{\mathbf{q}} = \mathbf{v}, \tag{5.106}$$
$$\dot{\mathbf{v}} = \mathbf{w}, \tag{5.107}$$
$$\mathbf{0} = \mathbf{M}(\mathbf{q})\mathbf{w} - \mathbf{f}(\mathbf{v}, \mathbf{q}, u) + \mathbf{G}^{\mathsf{T}}(\mathbf{q})\boldsymbol{\lambda}, \tag{5.108}$$
$$\mathbf{0} = \mathbf{g}_{qq}(\mathbf{q})(\mathbf{v}, \mathbf{v}) + \mathbf{G}(\mathbf{q})\mathbf{w}. \tag{5.109}$$

Since the original problem was index three, some care must be exercised when defining initial conditions to ensure that they are consistent. First, the initial conditions must satisfy (5.103), so following [68] we choose one of the position variables $\Theta(0) = 0$ and then compute the remaining six such that (5.103) is satisfied. The velocity-level constraint $\mathbf{0} = \mathbf{G}(\mathbf{q})\mathbf{v}$ is satisfied if we put $\mathbf{v}(0) = \mathbf{0}$. Finally, it remains to specify initial values for the algebraic variables $\mathbf{w}$, $\boldsymbol{\lambda}$, and u such that (5.108) and (5.109) hold. Rewriting these equations gives

$$\begin{bmatrix} \mathbf{M}(\mathbf{q}) & \mathbf{G}^{\mathsf{T}}(\mathbf{q}) \\ \mathbf{G}(\mathbf{q}) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f}(\mathbf{v}, \mathbf{q}, u) \\ -\mathbf{g}_{qq}(\mathbf{q})(\mathbf{v}, \mathbf{v}) \end{bmatrix}. \tag{5.110}$$

Thus, for given values of the position $\mathbf{q}$, velocity $\mathbf{v}$, and control u, we can solve for the corresponding values of $\mathbf{w}$ and $\boldsymbol{\lambda}$. The initial values computed this way are consistent and *for a constant* u are sufficient to completely determine the solution to the IVP. However, when the torque u is allowed to vary with time, there are many solutions and so, for comparison, we will impose the boundary condition $\beta(t_F) = q_1(t_F) = 15.8106$ rad. This specified value for the angle β is the same as the final value achieved when a constant torque is used. In other words, we will match the final state for the constant-torque IVP solution with the optimal control solution. Thus, the goal of the optimal control is to reach the same state in the same time while expending less "energy" (5.99).

The direct transcription method requires an initial guess for the state and control time functions. For most applications (including all of the other examples in this book), it suffices to supply a linear initial guess. In fact, the default procedure used by SOCS to compute the state at grid point k is

$$\mathbf{y}_k = \mathbf{y}_1 + \frac{(k-1)}{(M-1)}(\mathbf{y}_M - \mathbf{y}_1). \tag{5.111}$$

The user must specify the state at the initial grid point, $\mathbf{y}_1$, the state at the final grid point, $\mathbf{y}_M$, the initial and final times, t_I and t_F, and the number of grid points, M. A similar expression is used to define the control $\mathbf{u}_k$ and the grid time values t_k. However, it is often possible to construct a much better initial guess for the state and control variables using special information about the problem and, indeed, this is so for this multibody example. One obvious approach is to fix $u(t) = u_0$ and then integrate the index-one DAE system (5.106)–(5.109) using software for solving a DAE IVP such as DASSL [88, 89] or RADAU5 [68]. The second approach (also tested in [68]) is to apply an ODE method to the differential equations (5.106), (5.107). This technique requires an explicit expression for the vector $\mathbf{w}$ appearing on the right-hand side of (5.107). Fortunately, by solving (5.110),

$$\begin{bmatrix} \mathbf{w} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{M}(\mathbf{q}) & \mathbf{G}^{\mathsf{T}}(\mathbf{q}) \\ \mathbf{G}(\mathbf{q}) & \mathbf{0} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f}(\mathbf{v}, \mathbf{q}, u_0) \\ -\mathbf{g}_{qq}(\mathbf{q})(\mathbf{v}, \mathbf{v}) \end{bmatrix}, \tag{5.112}$$

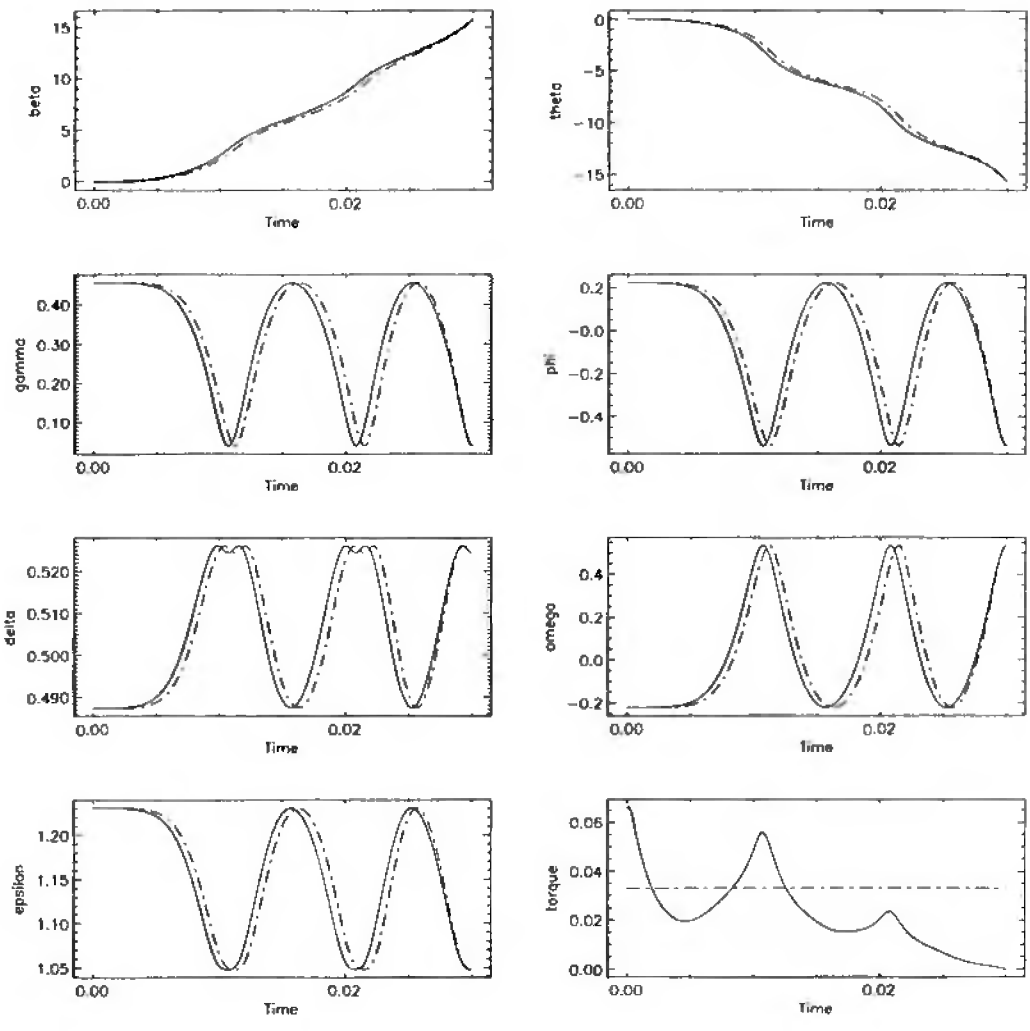


Figure 5.23: Squeezer mechanism solution.

one obtains the requisite explicit expression for $\mathbf{w} = \mathbf{w}(\mathbf{v}, \mathbf{q}, u_0)$. Of course, in practice we just solve (5.110) and do not explicitly compute the matrix inverse. An Adams predictor-corrector method was used to integrate the resulting ODE system, although any other IVP method could be used.

Figure 5.23 illustrates the solution obtained using SOCS. The minimum energy results are plotted with a solid line and the constant-torque reference trajectories using a dashed line. The minimum energy $J^* = 0.6669897295$ compared with a value $J = 1$ for the constant-torque reference trajectory. The initial guess was constructed by evaluating the integrated profile at 20 equally spaced grid points. The final solution was obtained after 4 mesh-refinement iterations; the algorithm performance is summarized in Table 5.13. It is interesting to note that the preferred discretization selected by the mesh-refinement algorithm, as described in Section 4.6.9, was the HSS (HS). To more fully appreciate this behavior, the same problem was solved 4 different ways—using 2 different

Iter.	Disc.	M	GE	HE	FE	RHS	$\epsilon_{\max}$	CPU (sec)
1	HS	20	29	15	2585	100815	0.66×10^{-4}	0.15×10^2
2	HS	39	18	15	2381	183337	0.98×10^{-5}	0.22×10^2
3	HS	72	10	7	1154	165022	0.27×10^{-5}	0.18×10^2
4	HS	143	7	2	456	129960	0.24×10^{-7}	0.20×10^2
Total		143	64	39	6576	579134		74.15

Table 5.13: Minimum energy squeezer mechanism.

	A	B	C	D
Index sets	16	28	38	70
FE per Hess./Jac.	152	434	779	2555
Total RHS eval.	579134	1628010	2879084	8399250
Largest NLP	7980	9044	5992	6790
Mesh-ref. iter.	4	5	4	5
Total CPU time	74.15	179.84	151.43	400.37
% Time for FE	23%	34%	55%	70%

Table 5.14: Discretization performance comparison.

discretizations, with and without right-hand side sparsity. For the sake of discussion, we refer to these as Methods A–D and they are summarized below:

Method	Discretization	Separability	RHS sparsity
A	HS	Yes	Yes
B	HS	Yes	No
C	HC	No	Yes
D	HC	No	No

An examination of the data in Table 5.14 explains why method A (HSS exploiting right-hand side sparsity) is the preferred approach. First, the number of index sets was 16. This can be attributed both to the separability of the discretization and the right-hand side sparsity as illustrated by the template (cf. (4.116))

$$\mathcal{T} = \text{struct}\begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \mathbf{y}} & \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{g}}{\partial \mathbf{y}} & \frac{\partial \mathbf{g}}{\partial \mathbf{u}} \\ \frac{\partial \mathbf{w}}{\partial \mathbf{y}} & \frac{\partial \mathbf{w}}{\partial \mathbf{u}} \end{bmatrix} = \begin{bmatrix} \times & & & & & & & & \\ & \times & & & & & & & \\ & & \times & & & & & & \\ & & & \times & & & & & \\ & & & & \times & & & & \\ & & & & & \times & & & \\ & & & & & & \times & & \\ & & & & & & & \times & \\ & & & & & & & & \times \\ \times\times & & \times\times & & \times\times & & \times\times\times\times\times\times\times & \\ \times\times & & \times & & \times\times & & \times\times\times\times\times\times & \\ & \times & & & & \times & & \times\times & \\ & \times\times & & \times & & \times\times & & \times\times & \\ & \times\times & \times\times & & \times\times & & \times\times & & \\ & & \times\times & & \times & & \times\times & & \times\times \\ & & \times\times & & \times\times & & \times\times & & \times\times \\ \times\times\times & & \times\times\times & & \times\times\times & & & & \\ \times\times\times & & \times\times\times & & \times\times\times & & & & \\ \times\times & \times\times & \times\times & \times\times & \times\times & \times\times & & & \\ \times\times & \times\times & \times\times & \times\times & \times\times & \times\times & & & \\ \times\times & & \times\times\times\times & & \times\times\times\times & & \times\times & & \\ \times\times & & \times\times\times\times & & \times\times\times\times & & \times\times & & \\ & & & & & & & & \times \end{bmatrix}. \tag{5.113}$$

When right-hand side sparsity is not exploited (Method B), the number of index sets increases to 28. Furthermore, when discretization separability is not exploited, the number

of index sets becomes larger still. Because the number of index sets for the HSS method is much smaller, the number of function evaluations needed to compute the Jacobian and Hessian is also significantly less. The net result is that the right-hand sides of the DAEs were evaluated a significantly smaller number of times. Notice that the final NLP was larger for the HSS discretization (Methods A and B). Nevertheless, method A is over 5.4 times faster than the HSC discretization without right-hand side sparsity (Method D). In simple terms, for this problem it is better to solve a larger NLP problem because the derivatives can be computed more efficiently!

The solution to this problem exhibits another phenomenon associated with the numerical solution of high-index DAEs. Figure 5.24 plots the time history of the errors in the path constraints. Specifically, it displays the acceleration-level error $\|\mathbf{g}_{qq}(\mathbf{q})(\mathbf{v},\mathbf{v}) + \mathbf{G}(\mathbf{q})\mathbf{w}\|$, the velocity-level error $\|\mathbf{G}(\mathbf{q})\mathbf{v}\|$, and the position error $\|\mathbf{g}(\mathbf{q})\|$. The acceleration error is acceptably small: $\|\frac{d^2\mathbf{g}(t)}{dt^2}\| \sim \mathcal{O}(10^{-11})$. However, both the position and velocity errors "drift" significantly from zero.

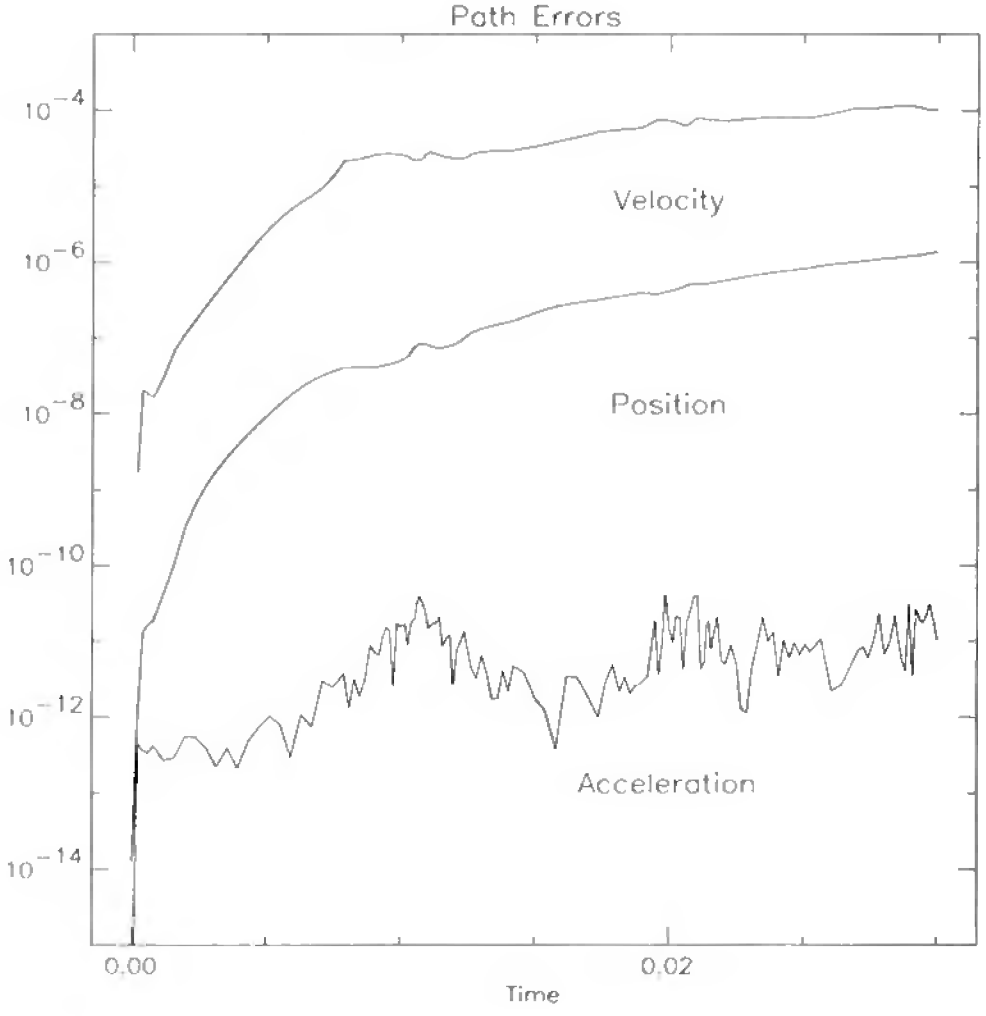


Figure 5.24: Path-constraint errors.

This page intentionally left blank

Appendix

Software

All of the algorithms described in the book have been implemented and tested on realistic test problems. In addition, all of the examples presented have been solved using this software, and FORTRAN implementations of the examples can be obtained by contacting the author. The SOCS [25] library, which contains all of the software, has a great deal of functionality and it is not intended that this appendix should be viewed as a user's manual. On the other hand, it is worthwhile to outline the basic capability of the tool to assist the reader in the formulation and solution of practical problems. To that end, the following sections present a brief overview of the library. Additional information can be obtained by contacting the author at `John.T.Betts@boeing.com` or from the SOCS webpage at `http://www.boeing.com/assocproducts/socs/`.

A.1 Simplified Usage Dense NLP

The subroutine IIDNLPD provides the most basic functionality for solving small, dense NLP problems. The primary information that must be supplied by the user (outlined with a double box in Figure A.1) is a subroutine called FUNBOX that evaluates the objective and constraint functions. This is the *function generator* described in Section 3.8. All algorithm parameters are given default values, which are appropriate for a simple problem. If desired, the default values can be redefined using a utility routine IIHSNLP that is common to all software in the SOCS library. The simplified usage software uses a *forward-communication* format and, as such, the optimization algorithm HDNLPD calls the user-supplied subroutine, which has a fixed calling argument list. For more sophisticated applications, the *reverse-communication* subroutine HDNLPR can be used. HDNLPR is appropriate for small, dense applications when the user can supply gradient and Hessian information. It is also appropriate for use with complicated simulation programs and when parallel processing is used.

A.2 Sparse NLP with Sparse Finite Differences

Large, sparse NLP problems necessarily demand more information from the user because of the need to specify matrix sparsity and provide corresponding derivative information. Figure A.2 illustrates the usage of the sparse NLP algorithm HDSNLP, which employs a reverse-communication format. Again, the user-supplied software is shown inside a

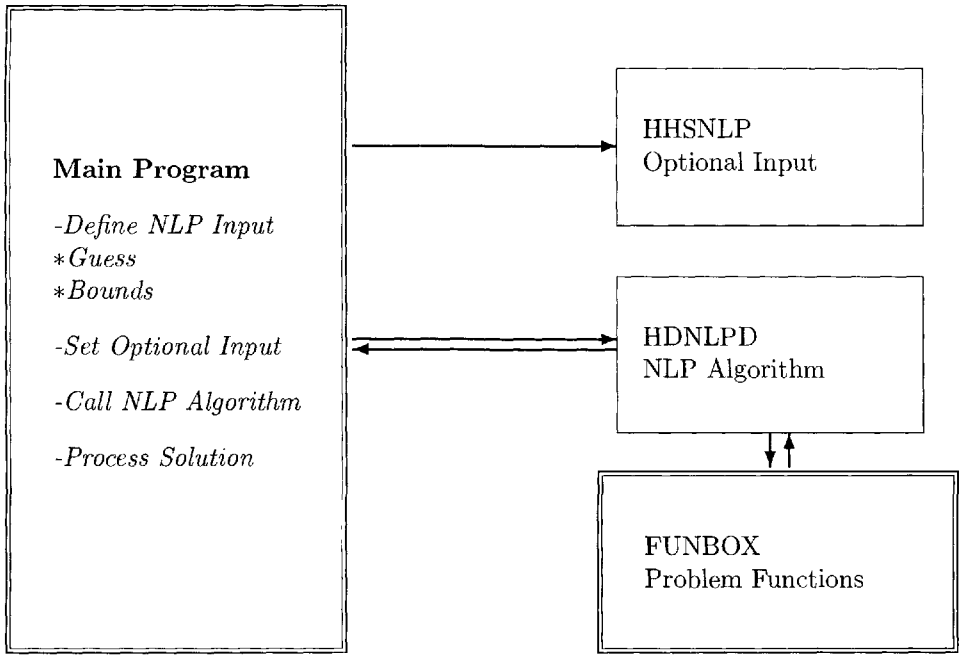


Figure A.1: Dense NLP software.

double box. In addition, software for constructing sparse finite difference first and second derivatives HDSFDJ/HDSFDH is also available. A utility procedure (HJSFDI) can be used for constructing sparse difference index sets based on the user-supplied matrix sparsity patterns. Optional input can be set using the standard SOCS utility HHSNLP. Similar functionality is available when solving large, sparse nonlinear least squares problems using subroutine HDSLSQ rather than HDSNLP.

A.3 Optimal Control Using Sparse NLP

Implementing the solution of an optimal control problem using the direct transcription method in SOCS requires some software supplied by the user. Figure A.3 illustrates the software organization of an application. As before, user-supplied procedures are shown with double boxes. However, many of the user-supplied routines are optional, as indicated by an asterisk in the illustration. The user must call the SOCS algorithm HDSOCS. The user must define the problem via the subroutine ODEINP. All other information is optional and can be supplied either by the user or by using dummy routines from the SOCS library instead. Typically, the user will specify the right-hand sides of the DAEs and quadrature functions using subroutine ODERHS. For applications with nonlinear boundary conditions, the point function routine ODEPTF must be supplied. If special output (e.g., graphics files) is desired, the user can supply a special-purpose ODEPRT routine. The default initial guess for the state and control variables is a linear function between the phase boundaries. When other initialization procedures are used, subroutine ODEIGS provides this functionality. When solving an optimal control problem, *it is not necessary* to define the matrix sparsity, to compute derivatives, or to call the sparse NLP

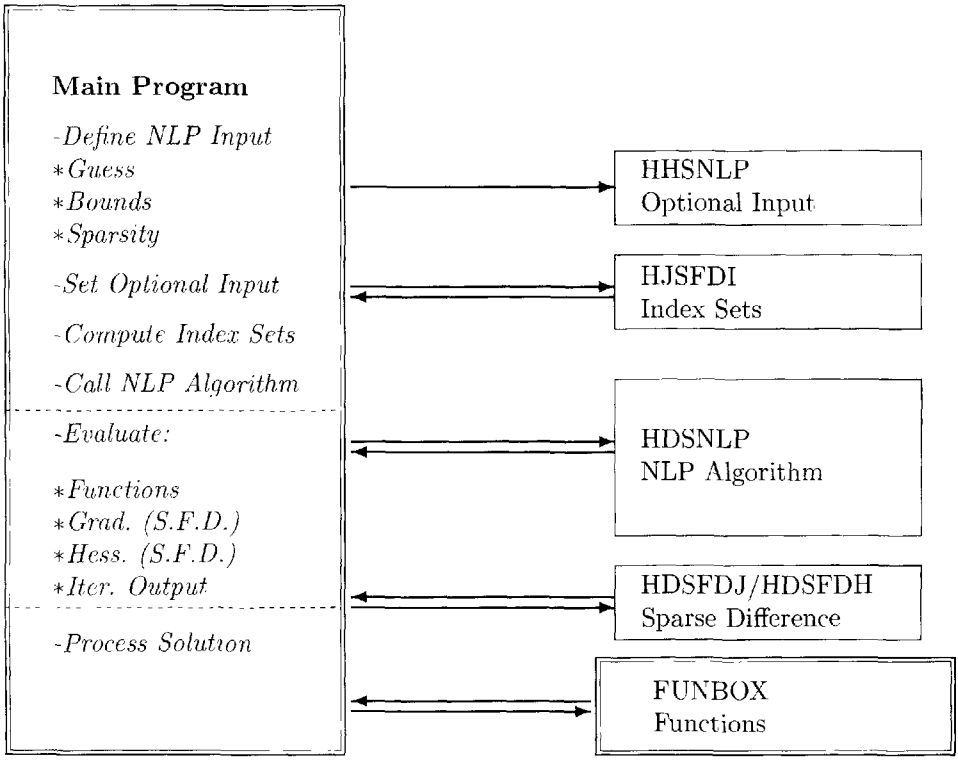


Figure A.2: Sparse NLP software.

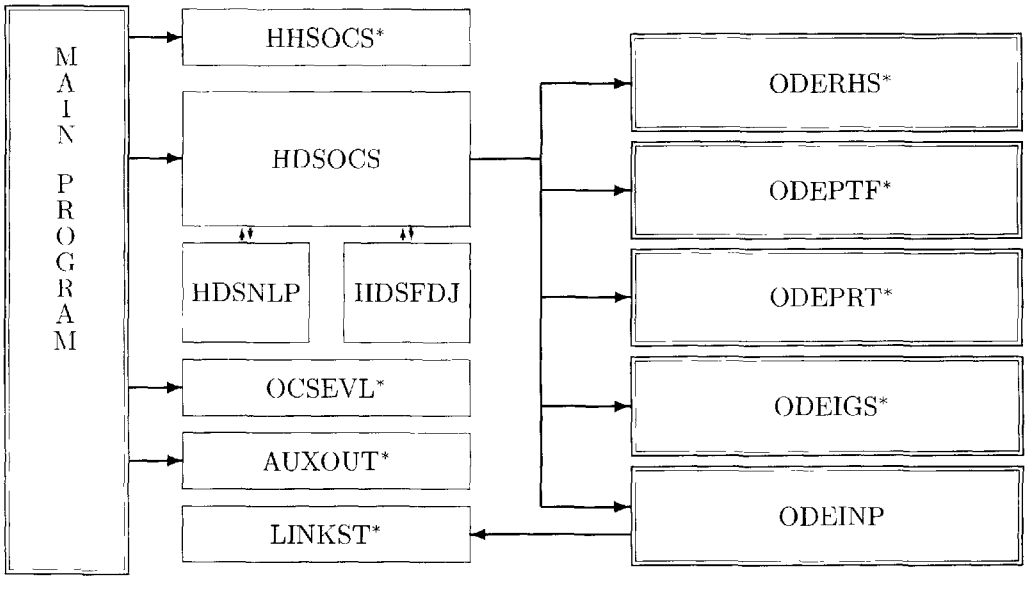


Figure A.3: Sparse optimal control software.

algorithm. However, nonstandard NLP options can be set using the HHSNLP utility. Nonstandard optimal control algorithm options can be set using the HHSOCS utility. For example, the discretization accuracy and number of mesh-refinement iterations can be set by HHSOCS. A feasible (but suboptimal) trajectory can be computed by using the feasibility (F) option as set by HHSNLP. The solution computed by SOCS is represented using B-spline function(s). The solution can be evaluated at arbitrary points with the utility OCSEVL and/or the auxiliary output procedure AUXOUT. Multiphase formulations may also incorporate the LINKST utility when linking phases together. The overall SOCS library has been designed with flexibility and functionality in mind and is especially suited for use with complex simulation systems.

Bibliography

[1] S. M. ALESSANDRINI, *A Motivational Example for the Numerical Solution of Two-Point Boundary-Value Problems*, SIAM Review, 37 (1995), pp. 423–427.

[2] U. M. ASCHER, R. M. M. MATTHEIJ, AND R. D. RUSSELL, *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[3] C. ASHCRAFT, R. G. GRIMES, AND J. G. LEWIS, *Accurate Symmetric Indefinite Linear Equation Solvers*, SIAM Journal of Matrix Analysis and Applications, 20 (1999), pp. 513 561.

[4] F. BASHFORTH AND J. C. ADAMS, *Theories of Capillary Action*, Cambridge University Press, New York, NY, 1883.

[5] R. H. BATTIN, *An Introduction to the Mathematics and Methods of Astrodynamics*, AIAA Education Series, American Institute of Aeronautics and Astronautics, Inc., 1633 Broadway, New York, NY 10019, 1987.

[6] P. BERKMANN AND H. J. PESCH, *Abort Landing in Windshear: Optimal Control Problem with Third-Order State Constraint and Varied Switching Structure*, Journal of Optimization Theory and Applications, 85 (1995), pp. 21 57.

[7] J. T. BETTS, *An Improved Penalty Function Method for Solving Constrained Parameter Optimization Problems*. Journal of Optimization Theory and Applications, 16 (1975), pp. 1 24.

[8] J. T. BETTS, *Solving the Nonlinear Least Square Problem: Application of a General Method*. Journal of Optimization Theory and Applications, 18 (1976), pp. 469-483.

[9] J. T. BETTS, *Using Sparse Nonlinear Programming to Compute Low Thrust Orbit Transfers*. The Journal of the Astronautical Sciences, 41 (1993), pp. 349 371.

[10] J. T. BETTS, *The Application of Sparse Least Squares in Aerospace Design Problems*, in Optimal Design and Control, Proceedings of the Workshop on Optimal Design and Control, J. Borggaard, J. Burkardt, M. Gunzburger, and J. Peterson, eds., Vol. 19 of Progress in Systems and Control Theory, Birkhäuser, Basel, Switzerland, Apr. 1994, pp. 81 96.

[11] J. T. BETTS, *Optimal Interplanetary Orbit Transfers by Direct Transcription*, The Journal of the Astronautical Sciences, 42 (1994), pp. 247–268.

[12] J. T. BETTS, *The Application of Optimization Techniques to Aerospace Systems*, in Fourth International Conference on Foundations of Computer-Aided Process Design, Proceedings of the Conference at Snowmass, CO, July 10–14, 1994, L. T. Biegler and M. F. Doherty, eds., CACHE, American Institute of Chemical Engineers, 1995.

[13] J. T. BETTS, *Survey of Numerical Methods for Trajectory Optimization*, AIAA Journal of Guidance, Control, and Dynamics, 21 (1998), pp. 193–207.

[14] J. T. BETTS, *Very Low Thrust Trajectory Optimization*, in High Performance Scientific and Engineering Computing, Proceedings of the International FORTWIHR Conference on HPSEC, Munich, March 16–18, 1998, H.-J. Bungartz, F. Durst, and C. Zenger, eds., Springer-Verlag, Berlin, Heidelberg, 1999.

[15] J. T. BETTS, *Very Low Thrust Trajectory Optimization Using a Direct SQP Method*, Journal of Computational and Applied Mathematics, 120 (2000), pp. 27–40.

[16] J. T. BETTS, N. BIEHN, S. L. CAMPBELL, AND W. P. HUFFMAN, *Exploiting Order Variation in Mesh Refinement for Direct Transcription Methods*, Tech. Report M&CT-TECH-99-022, Mathematics and Computing Technology, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, Oct. 1999.

[17] J. T. BETTS, N. BIEHN, S. L. CAMPBELL, AND W. P. HUFFMAN, *Compensation for Order Variation in Mesh Refinement for Direct Transcription Methods*, Journal of Computational and Applied Mathematics, 125 (2000), pp. 147–158.

[18] J. T. BETTS, M. J. CARTER, AND W. P. HUFFMAN, *Software for Nonlinear Optimization*, Mathematics and Engineering Analysis Library Report MEA-LR-83 R1, Boeing Information and Support Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, June 1997.

[19] J. T. BETTS, S. K. ELDERSVELD, AND W. P. HUFFMAN, *A Performance Comparison of Nonlinear Programming Algorithms for Large Sparse Problems*, in Proceedings of the AIAA Guidance, Navigation, and Control Conference, AIAA-93-3751-CP, Monterey, CA, Aug. 1993, pp. 443–455.

[20] J. T. BETTS AND P. D. FRANK, *A Sparse Nonlinear Optimization Algorithm*, Journal of Optimization Theory and Applications, 82 (1994), pp. 519–541.

[21] J. T. BETTS AND W. P. HUFFMAN, *Trajectory Optimization on a Parallel Processor*, AIAA Journal of Guidance, Control, and Dynamics, 14 (1991), pp. 431–439.

[22] J. T. BETTS AND W. P. HUFFMAN, *Application of Sparse Nonlinear Programming to Trajectory Optimization*, AIAA Journal of Guidance, Control, and Dynamics, 15 (1992), pp. 198–206.

[23] J. T. BETTS AND W. P. HUFFMAN, *Path Constrained Trajectory Optimization Using Sparse Sequential Quadratic Programming*, AIAA Journal of Guidance, Control, and Dynamics, 16 (1993), pp. 59–68.

[24] J. T. BETTS AND W. P. HUFFMAN, *Sparse Nonlinear Programming Test Problems (Release 1.0)*, BCS Technology Tech. Report BCSTECH-93-047, Boeing Computer Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, 1993.

[25] J. T. Betts and W. P. Huffman, *Sparse Optimal Control Software SOCS*, Mathematics and Engineering Analysis Tech. Document MEA-LR-085, Boeing Information and Support Services, The Boeing Company, PO Box 3707, Seattle, WA 98124-2207, July 1997.

[26] J. T. Betts and W. P. Huffman, *Mesh Refinement in Direct Transcription Methods for Optimal Control.* Optimal Control Applications and Methods, 19 (1998), pp. 1-21.

[27] J. T. Betts and W. P. Huffman, *Exploiting Sparsity in the Direct Transcription Method for Optimal Control*, Computational Optimization and Applications, 14 (1999), pp. 179 201.

[28] G. A. Bliss. *Lectures on the Calculus of Variations*, University of Chicago Press, Chicago, IL, 1946.

[29] K. E. Brenan, *Engineering Methods Report: The Design and Development of a Consistent Integrator/Interpolator for Optimization Problems*, Aerospace Tech. Memorandum ATM No. 88(9975)-52, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, 1988.

[30] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Vol. 14 of Classics in Applied Mathematics, SIAM, Philadelphia, PA, 1996.

[31] R. A. Broucke and P. J. Cefola, *On Equinoctial Orbit Elements*, Celestial Mechanics, 5 (1972), pp. 303–310.

[32] C. G. Broyden, *A Class of Methods for Solving Nonlinear Simultaneous Equations.* Mathematics of Computation, 19 (1965), pp. 577–593.

[33] C. G. Broyden. *The Convergence of a Class of Double-Rank Minimization Algorithms*, Journal of the Institute of Mathematics and Applications, 6 (1970), pp. 76–90.

[34] A. E. Bryson, Jr., M. N. Desai, and W. C. Hoffman, *Energy-State Approximation in Performance Optimization of Supersonic Aircraft*, Journal of Aircraft, 6 (1969), pp. 481–488.

[35] A. E. Bryson, Jr. and Y.-C. Ho, *Applied Optimal Control*, John Wiley & Sons, New York, NY. 1975.

[36] R. Bulirsch, *Die Mehrzielmethode zur numerischen Lösung von nichtlinearen Randwertproblemen und Aufgaben der optimalen Steuerung*, Report of the Carl-Cranz Gesellschaft, Carl-Cranz Gesellschaft, Oberpfaffenhofen, Germany, 1971.

[37] S. L. Campbell, N. Biehn, L. Jay, and T. Westbrook, *Some Comments on DAE Theory for IRK Methods and Trajectory Optimization*, Journal of Computational and Applied Mathematics, 120 (2000), pp. 109–131.

[38] M. D. Canon, C. D. Cullum, and E. Polak, *Theory of Optimal Control and Mathematical Programming.* McGraw–Hill. New York, NY. 1970.

[39] T. F. Coleman and J. J. Moré, *Estimation of Sparse Jacobian Matrices and Graph Coloring Problems*, SIAM Journal on Numerical Analysis, 20 (1983), pp. 187–209.

[40] A. R. Curtis, M. J. D. Powell, and J. K. Reid, *On the Estimation of Sparse Jacobian Matrices*, Journal of the Institute of Mathematics and Applications, 13 (1974), pp. 117–120.

[41] G. Dahlquist and Å. Björk, *Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, 1974.

[42] G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[43] W. C. Davidon, *Variable Metric Methods for Minimization*, A. E. C. Res. and Develop. Report ANL-5900, Argonne National Laboratory, Argonne, IL, 1959.

[44] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, NY, 1978.

[45] J. E. Dennis, Jr., D. M. Gay, and R. E. Welsch, *An Adaptive Nonlinear Least-Squares Algorithm*, Transactions of Math Software, 7 (1981), pp. 348–368.

[46] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983.

[47] E. D. Dickmanns, *Maximum Range Three-dimensional Lifting Planetary Entry*, Tech. Report TR R-387, National Aeronautics and Space Administration, 1972.

[48] T. N. Edelbaum, L. L. Sackett, and H. L. Malchow, *Optimal Low Thrust Geocentric Transfer*, in AIAA 10th Electric Propulsion Conference, AIAA 73-1074, Lake Tahoe, NV, Oct.–Nov. 1973.

[49] P. J. Enright and B. A. Conway, *Optimal Finite-thrust Spacecraft Trajectories Using Collocation and Nonlinear Programming*, AIAA Journal of Guidance, Control, and Dynamics, 14 (1991), pp. 981–985.

[50] P. J. Enright and B. A. Conway, *Discrete Approximations to Optimal Trajectories Using Direct Transcription and Nonlinear Programming*, AIAA Journal of Guidance, Control, and Dynamics, 15 (1992), pp. 994–1002.

[51] P. R. Escobal, *Methods of Orbit Determination*, Second Edition, Robert E. Krieger Publishing Company, Malabar, FL, 1985.

[52] R. Fletcher, *A New Approach to Variable Metric Algorithms*, Computer Journal, 13 (1970), pp. 317–322.

[53] R. Fletcher, *Practical Methods of Optimization, Vol. 2, Constrained Optimization*, John Wiley & Sons, New York, NY, 1985.

[54] R. Fletcher and S. Leyffer, *Nonlinear Programming without a Penalty Function*, University of Dundee Numerical Analysis Report NA/171, University of Dundee, Department of Mathematics, Dundee, DD1 4HN, Scotland, U.K., 1997.

[55] R. Fletcher and M. J. D. Powell, *A Rapidly Convergent Descent Method for Minimization*, Computer Journal, 6 (1963), pp. 163–168.

[56] C. W. GEAR, *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1971.

[57] C. W. GEAR, *The Simultaneous Numerical Solution of Differential-Algebraic Equations*, IEEE Transactions on Circuit Theory, CT-18 (1971), pp. 89–95.

[58] C. W. GEAR AND T. V. VU, *Smooth Numerical Solutions of Ordinary Differential Equations*, in Proceedings of the Workshop on Numerical Treatment of Inverse Problems for Differential and Integral Equations, Heidelberg, 1982.

[59] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *Some Theoretical Properties of an Augmented Lagrangian Merit Function*, Tech. Report SOL 86-6, Department of Operations Research, Stanford University, Apr. 1986.

[60] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *User's Guide for NPSOL (version 4.0): A Fortran Package for Nonlinear Programming*, Tech. Report SOL 86-2, Department of Operations Research, Stanford University, Jan. 1986.

[61] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *A Schur-complement Method for Sparse Quadratic Programming*, Tech. Report SOL 87-12, Department of Operations Research, Stanford University, Oct. 1987.

[62] P. E. GILL, W. MURRAY, M. A. SAUNDERS, AND M. H. WRIGHT, *A Schur-complement Method for Sparse Quadratic Programming*, in Reliable Numerical Computation, M. G. Cox and S. Hammarling, eds., Oxford University Press, Oxford, U.K., 1990, pp. 113–138.

[63] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Practical Optimization*, Academic Press, London, 1981.

[64] P. E. GILL, W. MURRAY, AND M. H. WRIGHT, *Numerical Linear Algebra and Optimization*, Addison-Wesley, Redwood City, CA, 1991.

[65] D. GOLDFARB, *A Family of Variable Metric Methods Derived by Variational Means*, Mathematics of Computation, 24 (1970), pp. 23–26.

[66] N. I. M. GOULD, *On Practical Conditions for the Existence and Uniqueness of Solutions to the General Equulity Quadratic Programming Problem*, Mathematical Programming, 32 (1985), pp. 90–99.

[67] E. HAIRER, S. P. NORSETT, AND G. WANNER, *Solving Ordinary Differential Equations. I. Nonstiff Problems*, Springer-Verlag, New York, NY, 1993.

[68] E. HAIRER AND G. WANNER, *Solving Ordinary Differential Equations. II. Stiff and Differential-Algebraic Problems*, Springer-Verlag, New York, NY, 1996.

[69] S. M. HAMMES, *Optimization Test Problems*, Aerospace Tech. Memorandum ATM 89(4464-06)-12, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, 1989.

[70] C. R. HARGRAVES AND S. W. PARIS, *Direct Trajectory Optimization Using Nonlinear Programming and Collocation*, AIAA Journal of Guidance, Control, and Dynamics, 10 (1987), pp. 338–342.

[71] M. T. HEATH, *Numerical Methods for Large Sparse Linear Least Squares Problems*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 497–513.

[72] M. HEINKENSCHLOSS, *Projected Sequential Quadratic Programming Methods*, SIAM Journal on Optimization, 6 (1996), pp. 373–417.

[73] A. L. HERMAN AND B. A. CONWAY, *Direct Optimization using Collocation Based on High-Order Gauss-Lobatto Quadrature Rules*, AIAA Journal of Guidance, Control, and Dynamics, 19 (1996), pp. 592–599.

[74] A. C. HINDMARSH, *ODEPACK, A Systematized Collection of ODE Solvers*, in Scientific Computing, R. S. Stepleman et al., eds., North Holland, Amsterdam, 1983, pp. 55–64.

[75] W. HOCK AND K. SCHITTKOWSKI, *Test Examples for Nonlinear Programming Codes*, Springer-Verlag, New York, NY, 1981.

[76] W. P. HUFFMAN, *An Analytic Propagation Method for Low Earth Orbits*, Internal Document, The Aerospace Corporation, 2350 E. El Segundo Blvd., El Segundo, CA 90245-4691, Nov. 1981.

[77] L. JAY, *Convergence of a Class of Runge-Kutta Methods for Differential-Algebraic Systems of Index* 2, BIT, 33 (1993), pp. 137–150.

[78] J. A. KECHICHIAN, *Equinoctial Orbit Elements: Application to Optimal Transfer Problems*, in AIAA/AAS Astrodynamics Specialist Conference, AIAA 90-2976, Portland, OR, Aug. 1990.

[79] J. A. KECHICHIAN, *Trajectory Optimization with a Modified Set of Equinoctial Orbit Elements*, in AIAA/AAS Astrodynamics Specialist Conference, AAS 91-524, Durango, CO, Aug. 1991.

[80] H. B. KELLER, *Numerical Methods for Two-Point Boundary Value Problems*, Blaisdell, Waltham, MA, London, 1968.

[81] C. T. KELLEY AND E. W. SACHS, *A Pointwise Quasi-Newton Method for Unconstrained Optimal Control Problems*, Numerische Mathematik, 55 (1989), pp. 159–176.

[82] C. T. KELLEY AND E. W. SACHS, *Pointwise Broyden Methods*, SIAM Journal on Optimization, 3 (1993), pp. 423–441.

[83] K. LEVENBERG, *A Method for the Solution of Certain Problems in Least Squares*, Quarterly of Applied Mathematics, 2 (1944), pp. 164–168.

[84] F. R. MOULTON, *New Methods in Exterior Ballistics*, University of Chicago Press, Chicago, IL, 1926.

[85] W. MURRAY AND M. H. WRIGHT, *Line Search Procedures for the Logarithmic Barrier Function*, SIAM Journal on Optimization, 4 (1994), pp. 229–246.

[86] M. OTTER AND S. TÜRK, *The DFVLR Models* 1 *and* 2 *of the Manutec r*3 *Robot*, DFVLR-Mitt. 88-3, Institut für Dynamik der Flugsysteme, Oberpfaffenhoffen, Germany, 1988.

[87] H. J. PESCH, *A Practical Guide to the Solution of Real-life Optimal Control Problems*, Control and Cybernetics, 23 (1994), pp. 7–60.

[88] L. R. PETZOLD, *Differential/Algebraic Equations Are Not ODEs*, SIAM Journal of Scientific and Statistical Computing, 3 (1982), pp. 367–384.

[89] L. R. PETZOLD. *A Description of DASSL: A Differential/Algebraic System Solver*, in Scientific Computing, R. S. Stepleman et. al., eds., North Holland, Amsterdam, 1983. pp. 65–68.

[90] E. POLAK, *Computational Methods in Optimization*, Academic Press, New York, NY, 1971.

[91] L. S. PONTRYAGIN, *The Mathematical Theory of Optimal Processes*, Wiley-Interscience, New York, NY, 1962.

[92] A. V. RAO AND K. D. MEASE, *Eigenvector Approximate Dichotomic Basis Method for Solving Hyper-Sensitive Optimal Control Problems*, Optimal Control Applications and Methods, 20 (1999), pp. 59–77.

[93] D. REDDING AND J. V. BREAKWELL, *Optimal Low-Thrust Transfers to Synchronous Orbit*. AIAA Journal of Guidance, Control, and Dynamics, 7 (1984), pp. 148–155.

[94] R. T. ROCKAFELLAR, *The Multiplier Method of Hestenes and Powell Applied to Convex Programming*, Journal of Optimization Theory and Applications, 12 (1973), pp. 555–562.

[95] G. SACHS AND K. LESCH, *Periodic Maximum Range Cruise with Singular Control*, AIAA Journal of Guidance, Control, and Dynamics, 16 (1993), pp. 790–793.

[96] W. SCHIEHLEN, *Multibody Systems Handbook*, Springer-Verlag, Berlin, Heidelberg, 1990.

[97] L. F. SHAMPINE AND M. K. GORDON, *Computer Solution of Ordinary Differential Equations: The Initial Value Problem*, W. H. Freeman, San Francisco, CA, 1975.

[98] D. F. SHANNO, *Conditioning of Quasi-Newton Methods for Function Minimization*, Mathematics of Computation, 24 (1970), pp. 647–657.

[99] J. L. SPEYER, *Periodic Optimal Flight*, AIAA Journal of Guidance, Control, and Dynamics. 19 (1996), pp. 745–755.

[100] J. STOER AND R. BULIRSCH, *Introduction to Numerical Analysis*, Springer-Verlag, New York, Berlin. Heidleberg, 1980.

[101] D. TABAK AND B. C. KUO, *Optimal Control by Mathematical Programming*, Prentice Hall, Englewood Cliffs, NJ, 1971.

[102] H. S. TSIEN AND R. C. EVANS, *Optimum Thrust Programming for a Sounding Rocket*. Journal of the American Rocket Society, 21 (1951), pp. 99–107.

[103] O. VON STRYK, *Numerical Solution of Optimal Control Problems by Direct Collocation*, in Optimal Control, R. Bulirsch, A. Miele, J. Stoer, and K. H. Well, eds., Vol. 111 of International Series of Numerical Mathematics, Birkhäuser-Verlag, Basel, Switzerland, 1993, pp. 129 143.

[104] O. VON STRYK AND R. BULIRSCH, *Direct and Indirect Methods for Trajectory Optimization*, Annals of Operations Research, 37 (1992), pp. 357–373.

[105] O. VON STRYK AND M. SCHLEMMER, *Optimal Control of the Industrial Robot Manutec r3*, in Computational Optimal Control, R. Bulirsch and D. Kraft, eds., Vol. 115 of International Series of Numerical Mathematics, Birkhäuser-Verlag, Basel, Switzerland, 1994, pp. 367–382.

[106] T. V. VU, *Numerical Methods for Smooth Solutions of Ordinary Differential Equations*, Tech. Report No. UIUCDCS-R-83-1130, Dept. of Computer Science, University of Illinois, Urbana, IL, May 1983.

[107] M. J. H. WALKER, B. IRELAND, AND J. OWENS, *A Set of Modified Equinoctial Orbit Elements*, Celestial Mechanics, 36 (1985), pp. 409–419.

[108] T. YEE AND J. A. KECHICHIAN, *On the Dynamic Modeling in Optimal Low-Thrust Orbit Transfer*, in AAS/AIAA Spaceflight Mechanics Meeting, AAS 92-177, Colorado Springs, CO, Feb. 1992.

[109] K. P. ZONDERVAN, T. P. BAUER, J. T. BETTS, AND W. P. HUFFMAN, *Solving the Optimal Control Problem Using a Nonlinear Programming Technique Part 3: Optimal Shuttle Reentry Trajectories*, in Proceedings of the AIAA/AAS Astrodynamics Conference, AIAA-84-2039, Seattle, WA, Aug. 1984.

[110] K. P. ZONDERVAN, L. J. WOOD, AND T. K. CAUGHY, *Optimal Low-Thrust, Three-Burn Transfers with Large Plane Changes*, The Journal of the Astronautical Sciences, 32 (1984), pp. 407–427.

Index